NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE

JOÃO PEDRO DE NORONHA FUNENGA
Undergraduate in Computer Science

PHYSICS-INSPIRED MACHINE LEARNING FOR ORBIT DETERMINATION IN LOW-EARTH ORBIT

MASTER IN ANALYSIS AND ENGINEERING OF BIG DATA

NOVA University Lisbon
August, 2023

DEPARTMENT OF
COMPUTER SCIENCE

PHYSICS-INSPIRED MACHINE LEARNING FOR ORBIT DETERMINATION IN LOW-EARTH ORBIT

JOÃO PEDRO DE NORONHA FUNENGA
Undergraduate in Computer Science

Adviser: Cláudia Alexandra Magalhães Soares
Assistant Professor, NOVA University Lisbon

Co-advisers: Marta Guimarães
Machine Learning Engineer, Neuraspace

Henrique Rego Costa
Flight Dynamics Engineer, Neuraspace

MASTER IN ANALYSIS AND ENGINEERING OF BIG DATA

NOVA University Lisbon
August, 2023

Physics-Inspired Machine Learning for orbit determination in Low-Earth Orbit

Acknowledgements

This project was an immensely challenging academic endeavor that tested my limits and pushed me beyond what I thought was possible. It was a team effort, and I want to extend my heartfelt appreciation to everyone who contributed to its success.

First and foremost, I want to thank my adviser, Cláudia Soares, who invited me to embark and venture on this endearing project. I am also grateful to my co-adviser, Marta Guimarães, who provided invaluable advice and support, making this work what it is. I also want to thank my temporary co-adviser, Henrique Costa, for his guidance and assistance in tackling this problem and for also passionately getting me into this topic. Furthermore, I am thankful to Neuraspace for providing me with the opportunity to conduct this research with them.

I owe my deepest gratitude to my closest family throughout this journey. They continuously reminded me of what life is all about and endured my countless practice presentations, even though most of them did not understand a word in English. To my mother, Teresa, the strongest person I know who ensured that I finished the dissertation and provided me the best emotional support I could ask for while facing her own challenges. To my father, Fernando, who helped me stay calm during stressful moments. To my grandparents, Libânia and António, who have always been my number one supporters and shaped the person I am today.

I would also like to express my gratitude to my closest friends, Catarina Almeida, João Brioso, Guilherme Garrillha, André Costa, Filipe Nunes, Martim Silva, Francisco Ramalho, Diogo Pinto, Francisco Caldas, Joel Santiago, and all the others who have supported me throughout my life. Each of you provided me with unconditional support in your unique ways, and I am forever grateful for your friendship.

Finally, I would like to thank music and the role it played as being my constant companion. Additionally, I am grateful for the friendships that have blossomed through our shared love for music and the support given by these relationships.

This research was carried out under Project "Artificial Intelligence Fights Space Debris" Nº C626449889-0046305 co-funded by Recovery and Resilience Plan and NextGeneration EU Funds, www.recuperarportugal.gov.pt.

“You can’t really start living until you can live with yourself.”
(Thebe Neruda Kgositsile)

Abstract

We have always been told since we were little that space is infinite. Having this in mind, it would not make much sense to be so cautious and aware of the space that lies above us. However, the area right above the Earth's surface up to 2000 km is heavily contaminated with space debris which can have all kinds of origins and dimensions both man-made (inactive satellites, parts of rockets, minuscule flecks of paint) as well as from natural sources (small meteoroids). Considering that satellites have their propellant carefully measured to fulfill the planned trajectory and cannot afford evasion maneuvers at the slightest danger signal, it is important to quantify the uncertainty on the predictions made.

To predict when two objects will collide, one will need to model their orbits with the goal of knowing their positions. Among the multiple elements involved, such as the gravity potential or the shape of the object, space weather is the most difficult to predict. Because of these stochastic variables, the early discoveries from multiple scientists in the eighteenth century were only enough to describe an orbit in the perfect case scenario. These variables make the modeling of a real orbit more challenging since they are random and have to be considered when modeling them since their effects are not negligible. One of the variables that has the most impact on calculating the orbit of a space object is atmospheric density. Since we are dealing with a physical system that abides by physical laws, even if not perfectly, this will be used to our advantage to improve the predictions made. As aforementioned, these laws known for centuries can be too tailored for the perfect-case scenario and new equations can be discovered to better model a real-case scenario. The objective of this research is to employ physically informed machine learning techniques for orbit determination as well as to model atmospheric density by leveraging physical domain knowledge and improving upon the standard approach.

Keywords: Physically-Informed Neural Networks, Data-driven physical discovery, Space Debris, Orbital Mechanics, Orbit determination

Resumo

Sempre nos foi dito desde pequenos que o espaço é infinito. Tendo isto em conta, não faria muito sentido ser tão cauteloso e consciente acerca do espaço que se encontra acima de nós. No entanto, a zona mesmo acima da superfície terrestre até 2000 km está bastante poluída com detritos espaciais que podem ter todo o tipo de origens e dimensões tanto de origem humana (satélites inactivos, pequenas partes de foguetes, lascas de tinta), bem como de fontes naturais (pequenos meteoroides). Considerando que os satélites têm o seu combustivel cuidadosamente medido para a trajectória que está planeada e não pode realizar manobras evasivas ao menor perigo, é importante quantificar a incerteza sobre as previsões feitas.

Para prever quando dois objectos irão colidir, será necessária a modelação das suas órbitas com o objectivo de conhecer as posições respetivas, cujo procedimento tem múltiplas variáveis envolvidas, sendo as mais difíceis de prever, as relacionadas com o tempo espacial. Estas variáveis estocásticas mudam a forma de como as descobertas de múltiplos cientistas do século XVIII descreverem como se definia uma órbita perfeita. Estas variáveis tornam a modelação de uma orbita real mais desafiadora uma vez que são aleatórias e têm de ser consideradas uma vez que os seus efeitos não são desprezáveis. Uma das variáveis que tem mais impacto no cálculo das órbitas é a densidade atmosférica.

Uma vez que estamos perante um sistema físico que obedece às leis físicas, mesmo que não perfeitamente, isto será utilizado a nosso favor para melhorar as previsões feitas. Como acima mencionado, estas leis conhecidas há séculos têm como função modelar o caso perfeito e novas equações podem ser descobertas para melhor representar um cenário real. O objectivo deste trabalho é, com a utilização de uma rede neuronal para prever a densidade atmosférica juntamente com o conhecimento do domínio físico, as previsões feitas serão melhoradas comparativamente a uma abordagem em que é desprezado o contexto do sistema em que se inserem.

Palavras-chave: Redes Neuronais fisicamente informadas, Descoberta de equações baseadas em dados, Lixo espacial, Mecânica Orbital

Contents

List of Figures

List of Tables

Glossary

orbital elements	A set of 6 elements that are needed to define a *Keplerian* Orbit 6
Principal Component Analysis	PCA is a statistical technique used for dimensionality reduction having the objective of transforming high-dimensional data into lower-dimensionsional while retaining as much information as possible. 16
state vector	A vector corresponding to the state of a satellite. It consists of the position and velocity values in each axis. Additionally, the acceleration values can be added to the vector. 2, 6, 22

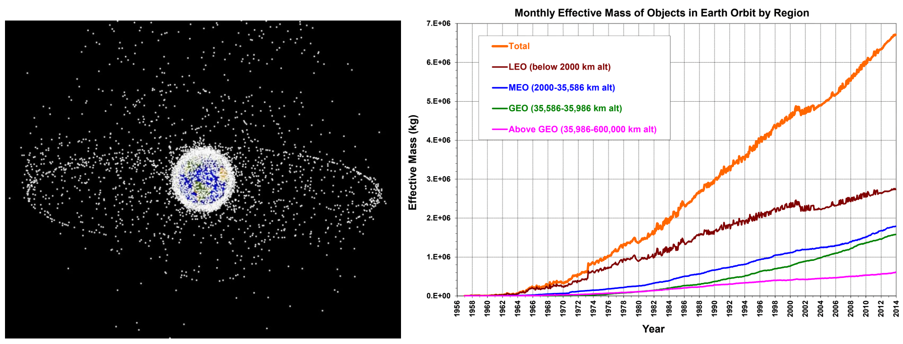
Acronyms

bPINNs	Bayesian Physically-informed Neural Networks 13
FROLS	Forward Regression Orthogonal Least-Squares 30
GANs	Generative adversarial networks 12
GEO	Geosynchronous Equatorial Orbit 1
HASDM	Air Force Space Battlelab's High Accuracy Satellite Drag Model 16
LEO	Low Earth Orbit 1, 2, 16, 17, 20, 21, 34
MEO	Medium Earth Orbit 1
ML	Machine Learning 4, 5, 10, 11, 13, 15, 16, 17
NN	Neural Network 5, 10, 11, 13, 16
PDE	Partial Differential Equation 5, 11, 12, 13, 14, 15, 18, 19, 28, 29, 31, 32, 35, 36, 38, 39, 70
PINNs	Physically-informed Neural Networks 4, 5, 11, 12, 13, 14, 15, 28, 46, 47, 61, 62, 65, 69, 70
RAAN	Right Ascension of the Ascending Node 23
RSO	Resident Space Object 20, 21
SINDy	Sparse identification of nonlinear dynamical systems 17, 18, 19, 20, 21, 28, 29, 30, 32, 34, 36, 38, 39, 62, 70
SSO	Sun-synchronous Orbit 21
SVM	Support Vector Machine 21

1

Motivation and Problem Statement

Space debris consists of objects orbiting the Earth. Sometimes this term might be interpreted as only man-made objects but official sources [3] state that it also includes natural debris like meteoroids. This results in a different term since most of the meteoroids are orbiting around the sun while most of the man-made debris orbits around the Earth, the latter being referred to as **orbital debris**. The three main space regions polluted with debris are Low Earth Orbit (LEO) which ranges from 200 km to 2000 km of altitude, Medium Earth Orbit (MEO) which corresponds to the region between LEO and Geosynchronous Equatorial Orbit (GEO), and finally GEO which has a constant altitude of around 36000 km as seen in Figure 1.1(a).



(a) Representation of space debris (b) Space debris mass distribution

Figure 1.1: Debris distribution - A depiction (not to scale) of the distribution of space debris and a plot corresponding to the distribution of the debris mass between different space regions over the years. Three main regions seen are LEO (cloud around the earth), MEO (region between LEO and GEO) and GEO (further away ring) respectively [2].

Orbital debris consists of man-made objects that no longer serve a useful purpose ranging from a whole non-functional spacecraft to fragments of destroyed satellites. One major event that notably increased the amount of space debris in LEO was the chinese anti-satellite missile test [74] whose objective was to be the first space mission since 1985 to intercept a satellite. This consisted of using another spacecraft as a kinetic weapon hitting the original satellite which resulted in around 3,000 pieces of trackable debris that ended up being in orbit in LEO and an estimated 32,000 smaller untracked pieces. The initial debris cloud generated was spread out over all of LEO after three years. Just two years later, there was an accidental collision between two communication satellites (Russian and American) in which the Russian satellite had already been deactivated previously to the impact that occurred at a velocity of 11.7 km/s [18]. This impact resulted in 2,000 pieces of debris, measuring at least ten centimeters in diameter, and many thousands of smaller pieces that will remain in orbit for years, posing a threat to other objects in LEO [26, 75]. This increases the importance of dealing with space debris such as this deactivated satellite and the dangers they still pose even when they are no longer being operated.

Having these pieces of debris in orbit freely sharing the same space with active spacecrafts worth millions of euros poses a big threat and is extremely risky for their safety. Collisions with debris, even of a small dimension, should be avoided, given that small objects in an orbit can reach speeds as high as 8 km/s in LEO with a similar effect to a bullet [44]. The design of the satellites take these hazards into consideration [45]. However, there are specific parts that cannot be easily protected and their damage can significantly hinder efficiency. This is the case of solar panels, for example, which when impacted may cause a disastrous electrical discharge that disrupts onboard systems [14, 4].

Focusing our attention now oo the active satellites, orbits cannot be perfectly defined by the seventeenth century *Keplerian* model. This model considers that the Earth is a perfect sphere, which is far from true due to its oblateness [12, section 4.7] and it considers that a satellite is only affected by the Earths gravity force. This model neglects any kind of gravitational interactions with other objects, atmospheric drag or solar radiation pressure.

One of the forces that impacts the positions of satellites the most is drag [67, 41]. If this is not taken into account, propagating the position of a satellite for the next 24 hours using such a simplified model can entail a large error. This is due to the fact that drag reduces the size of the orbit over time, meaning that the altitude of perigee will remain approximately constant but the altitude of apogee gets smaller [42]. If one discards these changes, the errors for the positions will increase continuously.

The satellites in orbit around the Earth have at least three major forces applied to them which are the **gravity** that pulls them downwards towards the Earth, **drag** which reduces their acceleration and acts in the opposite direction of the velocity, and the **propulsion** force which can be occasionally applied for corrective maneuvers, attitude control or a collision avoidance maneuver [1]. Knowing these forces, it is possible to solve a second order differential equation that describes the dynamics of the satellite. By integrating the acceleration we get the velocity of the satellite and integrating twice we get the position which means that by only knowing the acceleration it is possible to get the velocity and position. Having one state vector for a satellite and knowing which forces are applied, we can propagate the orbit, calculating its state vector in the future. Accurately determining the state vectors provides a comprehensive understanding of these forces and their impact on the trajectory of satellites. By considering these factors holistically, satellite operators can anticipate and mitigate their effects, ensuring the satellite stays on its intended path. The differential equation that governs the motion of satellites and is used for propagating the orbit is given by

$$\frac{d^2}{dt^2}\vec{r} = -\frac{\mu}{\|\vec{r}\|^3}\vec{r} - \vec{a}_{drag} - \vec{a}_{SRP}, \tag{1.1}$$

where μ is the Earth standard gravitational parameter, $\vec{r}$ is the position vector of the satellite relative to the center of the Earth, $\vec{a}_{drag}$ is the acceleration vector of drag and $\vec{a}_{SRP}$ is the solar radiation pressure applied on the satellite.

The presence of drag in equation (1.1) changes the nominal trajectory of the object, mainly reducing its altitude, which is why, as time goes by, maneuvers need to be performed and, in some cases, propellant needs to be used to get the satellites back to their nominal altitude. The drag, related to the velocity and physical properties of the satellite, is described as

$$\vec{a}_{drag} = \frac{1}{2}\rho C_D \frac{A}{m}\|\dot{\vec{r}}\|\dot{\vec{r}}, \tag{1.2}$$

where ρ corresponds to the atmospheric density, C_D is the drag coefficient of the satellite, A is the reference area of the satellite, m is the mass of the satellite and $\dot{\vec{r}}$ the velocity vector of the satellite.

All of the terms in (1.2) are rather easily measured or calculated apart from one which has the most uncertainty associated with its predictions, the atmospheric density [41]. This real positive variable is a function of the position in space and time. Atmospheric density is stochastic and cannot be easily determined for a timestamp in the future. This variable has a regular behavior in certain conditions which is currently well modeled. One example is the following, the space that is turned towards the Sun (during the day, between the Sun and the Earth) compared to the space on the other side of the Earth (during the night) has a higher density because the atmosphere receives more energy from the Sun and is denser at higher altitudes [61]. However, there is a phenomenon that is still not well modeled and affects this property of the atmosphere. The Sun has cycles with a period of about eleven years and consist on movements of masses of magnetically charged gas that ultimately causes a flip on the solar magnetic field, with the North and South poles exchanging places. The solar cycle features movements of large solar activity — solar flares — that can cause impacts on Earth, e.g., on the electric grid [16] or even on the positions of satellites themselves [32, 24, 53].

As the solar cycle progresses (as seen in 1.2), the amount of activity also varies which results in more solar explosions. The beginning of the cycle is a solar minimum in which the activity is smaller and at the middle of the cycle, the Sun reaches its solar maximum in which there are more solar explosions. This solar activity naturally affects the atmospheric density but unfortunately these explosions are difficult to predict [35] and this results in added uncertainty about the position of the satellites which in the worst case scenario can make spacecraft unusable.

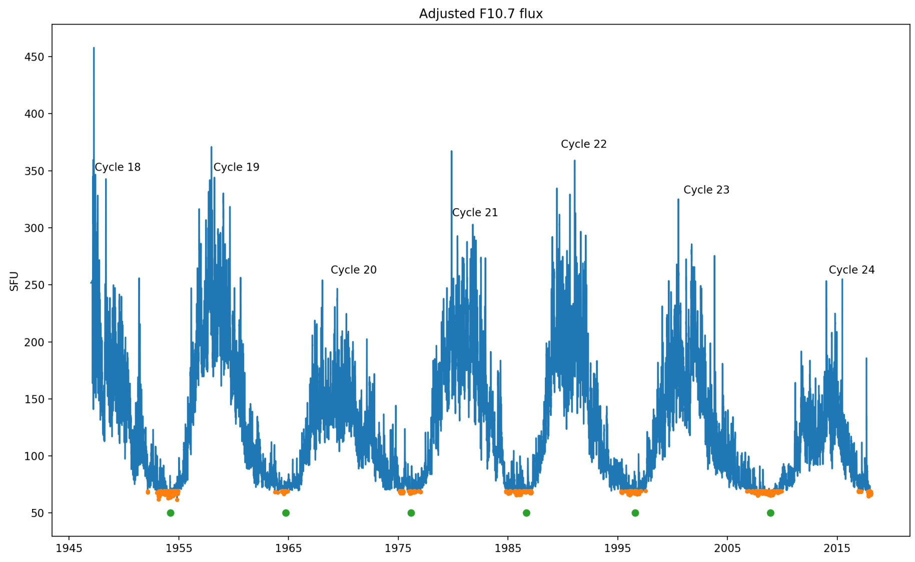


Figure 1.2: Solar Cycle and the evolution of the Solar Activity

It is important to note that in (1.1), apart from the drag term, there are other forces applied on a satellite for example the solar radiation pressure, its interaction with other celestial bodies or even interactions with other satellites. However, these interactions are often discarded due to the trade-off between the difficulty of calculating these extra terms and the decrease in performance by considering them in the equations. Since every object in space has a gravitational force applied from every other object this would end up being intractable. With this in mind, one approach we will follow is to use a technique called Sparse Identification of Non-Linear Dynamics (explained in 3.1.3) in order to uncover the governing equations of a system with an empirical approach having the possibility of discovering extra terms that are not in the known theoretical equations.

Physically-informed Machine Learning was recently created specifically to model difficult physical systems [30, 22, 49, 55]. Apart from the classical least squares regression methodology discovered in the end of the 1700's by Gauss to fit a line to observed data with the goal of making predictions for the future [66], Machine Learning (ML) has recently been having a lot of advances and its advantages are broadly talked about. Considering that one has a large enough training dataset, it is possible to find a model that accurately describes interactions between multiple variables which in turn helps on making better predictions. However, classical ML has one big problem in this context, it relies on having sufficiently large training datasets. This is where Physically-informed Neural Networks (PINNs) come in. These are a type of neural network that incorporate physical laws into the design and training of the network, more specifically on the optimization process. Since the model integrates the laws of the physical system, such constraints to the prediction space are thought to improve the predictions made, due to the knowledge that they must follow the laws that rule that physical system [37], even if not thoroughly. PINNs are trained by aggregating observed data from a specific system with known physical laws in order for its predictions to be consistent with the physics that rule that system.

With traditional numerical methods, solving real-world physics problems with data measured across different time intervals or noisy boundary conditions is impossible [31]. These kinds of methods are usually very rigorous and need a lot of assumptions that most of the time are impossible to guarantee in the real-world. This is where ML comes in hand, due to its ability to identify multidimensional correlations easily and solve ill-posed problems that would be intractable with traditional methods. This approach with Neural Network (NN) is efficient and simple [31], enabling difficult simulations at the expense of being harder to run. In a purely data-driven approach, it is possible to obtain models that are very accurate for the training data, but the predictions made may be physically inconsistent or implausible leading to a very poor generalization in the test data. Because of this flaw on the predictions by this type of model, it is necessary to add fundamental laws of physics and domain knowledge that serves as prior information and introduces bias (in addition to the inherent observed bias that is necessary for the model to learn assumptions). In summary, this means we are using prior physical and mathematical knowledge of the domain to help improve the performance of the model. The main motivation for building these types of algorithms is to have models that remain robust in the event of missing data or noisy values and can predict accurately and consistently by leveraging the physics of the system in which they are embedded. For complex physical systems, the amount of data that exists is limited given the complex behavior to be captured by the model so a purely data driven approach would not work.

The amount of existing data associated with the problem and the physical laws that are known to represent the system [31] can be seen as three categories of problems. On **one extreme** there is the case where little data is available (just the data for the initial and boundary conditions) to work with, but everything is known about the physics of the system (down to the coefficients of the partial derivative equations that define it). On the **other extreme**, one may not even know anything about the physics of the system but big data is available to use thus one should follow a data-driven approach in which advantage can be taken from the huge amount of data being worked with. In the **middle**, there is a balance between the two, in which there is some physical knowledge about the system (possibly with some values for missing parameters) and also some data to work with, this is where PINNs shine and where they are most useful to apply.

In this intermediate case when one can assume that there is partial knowledge of the physics, this usually corresponds to partially knowing the conservation laws which is not enough to derive the constitutive relationships. This kind of problem where we have a little bit of both can lead to very complex scenarios where the solution of the Partial Differential Equation (PDE) is a stochastic process due to external stimuli that can be random or an uncertain property of a material and therefore stochastic PDEs can be used to represent these stochastic solutions and uncertainties.

2

Orbital Dynamics

The field in which this problem is inserted has quite a lot of terminology and context that is crucial to understand before delving into the data analysis or what the state of the art is on approaching this problem.

2.1 Keplerian Orbit

The data that will be available for each satellite corresponds to the positions and velocities in a J2000 frame [48] and in a Cartesian referential for the three axes over time (state vector). From this information, it is possible to define an orbit in multiple formulations, the most common one being the Keplerian Orbit [12, section 4.4]. To define an orbit, there are 6 parameters which are also referenced as orbital elements [73].

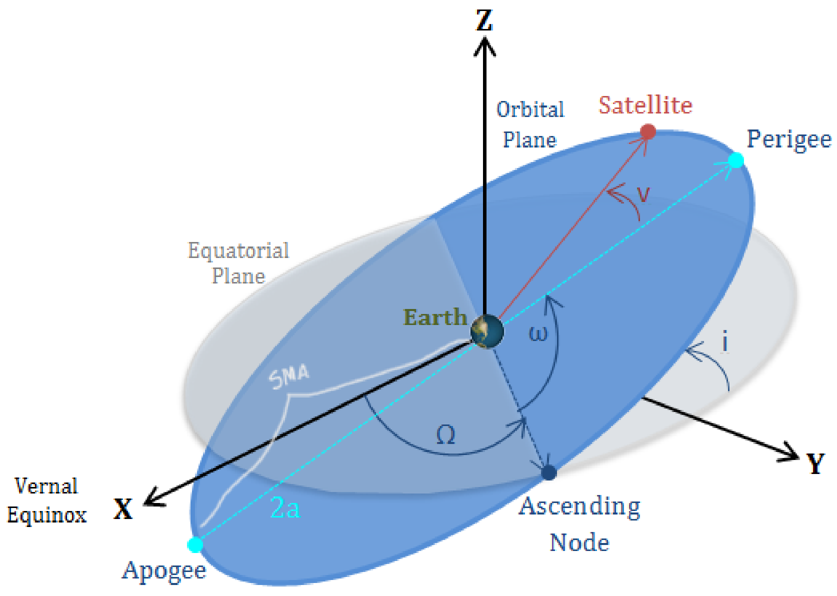


Figure 2.1: Visual representation of the orbital elements

Firstly, the **semimajor axis** (represented as SMA in 2.1) corresponds to the distance from the center of the orbit to the farthest point from the center (apogee). Secondly, the **eccentricity** is a ratio concerning the shape of an orbit. This parameter describes how much an orbit deviates from a perfect circle and it ranges from 0 to 1. This means that an orbit with eccentricity 1 is represented by a line and an orbit with eccentricity 0 corresponds to a perfect circle. This ratio is given by

$$eccentricity = \frac{r_{apogee} - r_{perigee}}{r_{apogee} + r_{perigee}},$$

where r_{apogee} corresponds to the distance between the point where the body is the farthest from the earth and the earth whereas $r_{perigee}$ corresponds to the distance between the point where it is the closest to earth and the earth.

The following parameter is the **inclination** (represented as i in 2.2) of the orbit which corresponds to the angle between the equatorial plane and the orbital plane, ranging from 0° to 180°.

The next parameter is the **right ascension of the ascending node** (represented as Ω in 2.1) which is a measure of the orientation of an orbit in space. It is defined as the angle between the direction of the vernal equinox, which is a fixed reference point in the sky and the point where the orbit of an object intersects the equatorial plane.

The following parameter, **argument of perigee** (represented as ω in 2.1), corresponds to the angle between the ascending node of the body to the perigee of the orbit which represents an arc. This means that if this argument is zero, the lowest point (altitude-wise) in the orbit of the object will correspond to the crossing of the equatorial plane of the celestial body it is revolving around.

This leaves us with one parameter left that does not define the orbit per se but is used to determine the position of the body in the orbit itself, the **true anomaly** (represented as v in 2.1).

With these parameters now known, it is possible to describe a Keplerian orbit. The position of a satellite is already given by the state vector that defines it but this new representation helps by giving a different perspective on how the trajectory of an object in an orbit is described. This representation is independent of the viewing perspective which means they can be used to predict the position from any point and represent a standardized way of looking at the motion of celestial objects making it easier to analyze and compare different orbits.

2.2 Oblateness

Due to the high rotational speeds, planets tend to bulge out around the equator because of centrifugal force [8]. Measuring the Earth, its equatorial radius is 21 km larger than the polar radius [12, section 4.7]. This flattening at the poles of the Earth is called oblateness and can be described by the ratio

$$oblateness = \frac{r_{equatorial} - r_{polar}}{r_{equatorial}}.$$

Lacking the symmetry that a sphere has, signifies that for an orbiting body around the Earth, the gravity force is not directed towards the center of the earth and depends both on the distance from its center and the latitude it is in, which can be seen as the angular distance from the equator, a phenomenon called **zonal variation** [12, section 4.7]. What is important to take from knowing that we are not dealing with the perfect case scenario is that this occurrence changes the orbit parameters over the time which in turn results in changing the orbit itself.

The average rate of change of precession of the orbital plane (node line) is given by $\dot{\Omega}$ which can be expressed as

$$\dot{\Omega} = -\left[\frac{3}{2}\frac{\sqrt{\mu}J_2R^2}{(1-e^2)^2a^{\frac{7}{2}}}\right]\cos i, \tag{2.1}$$

where μ is the gravitational parameter of the celestial body, J_2 is a dimensionless parameter that quantifies the effects of the oblateness of that celestial body on orbits, R is the radius of the celestial body, e is the eccentricity, a is the semi-major axis and i is the inclination of the orbit. Interpreting (2.1), it is possible to conclude that when the inclination of the orbit is between 0º and 90º, the rate of change of precession of the orbital plane is smaller than 0 which will drift posigrade orbits westwards. In case the inclination is between 90º and 180º, that rate is larger than 0 which means the orbits will drift in the opposite direction (eastwards). If the inclination of the orbit is exactly 90º, the orbit plane will not rotate.

The average rate of change of the argument of perigee is

$$\dot{\omega} = -\left[\frac{3}{2}\frac{\sqrt{\mu}J_2R^2}{(1-e^2)^2a^{\frac{7}{2}}}\right]\left(\frac{5}{2}\sin^2 i - 2\right).$$

This represents that if its inclination is between 0º and 63.4º or between 116.6º and 180º, the rate of change is positive which means the perigee advances in the direction the satellite is moving. If the inclination is between 63.4º and 116.6º the opposite happens, and the perigee regresses. Similarly to what happens with the node line, if the inclination is exactly 63.4º or 116.6º the perigee does not change position.

2.3 Current Atmospheric Density Modelling

Going back to our problem, there are currently two popular neutral density models that try to model the atmospheric density, **JB08** and **NRMLSIS** [6, 51, 76]. The former has an excellent calibration of densities concerning solar radiation although having a low global resolution since the temperature at the thermosphere base is fixed and the semi-annual variation is not calculated in a physically-informed manner. The latter has less variables but encapsulates more domain knowledge concerning the physical system leading to a more accurate model. The atmospheric density has two low peaks and two high peaks each year (Semi-Annual Variation) resulting in lower densities during the summer in the northern hemisphere summer compared to the summer in the southern hemisphere. Whereas JB08 uses an empirical adjustment to the densities after they are derived from the exospheric temperature, in NRLMSIS the exospheric temperatures are modified before deriving densities.

In Figure 2.2 that compares both of these models, it is possible to see that the error for JB08 is increasing exponentially whereas the error for NRMLSIS is not. However, it is possible to see that the error is larger for the latter up to the fifth day meaning one cannot be said to be better than the other, it is a matter of the use-case. After 7 days, the position calculated for a satellite can have an error of around 8 km using the JB08 model and of around 3 km for the NRLMSIS. This means that if an operator is trying to determine if there will be a collision between two objects in 7 days, if the error is 3 km for each of the bodies, both of them will have an accumulated error of 6 km which can be dangerous in critical applications such as in a collision avoidance scenario.

The **main goal** of this work is to develop a model which not only reduces the errors achieved by the state-of-art techniques but also change the exponential behaviour of the error.

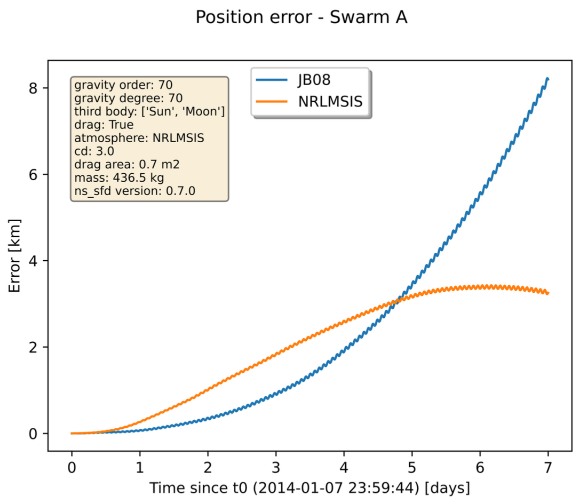


Figure 2.2: Error for the positions increases exponentially for JB08 although being smaller than with NRMLSIS for the first 4 days.

3 Physically-Inspired Machine Learning

As stated in the first chapter (1), Physically-Inspired Machine Learning has been gaining a lot of traction over the last few years. Using ML techniques coupled with physical knowledge helps to better model and predict variables in a physical system compared to a data-centric approach.

When building a Physically-Informed learning algorithm, certain biases need to be introduced to steer the learning process towards identifying physically consistent solutions. Some biases, such as observational biases, are inherently introduced, while others, including inductive and learning biases, can be enforced by us [31]. Firstly, **observational bias** is introduced through the data, which implicitly respects the underlying physics of the problem. Secondly, **inductive bias** corresponds to assumptions that can be incorporated into a NN (for example forcing the predictions to satisfy a set of physical laws), and thirdly, **learning bias** which can be introduced when choosing the loss function, using constraints on the NN or inference algorithms that may favor convergence to solutions that respect the implicit physical laws. By tuning these soft constraints, the physical laws can only be approximately respected, which increases the flexibility of implementing these kinds of laws. Using all these biases [49] together to create a physically consistent system is a good approach and it can be done in the following way:

- **Observational biases** - are the main pillar of advances in ML and the easiest way to introduce biases in ML. With enough data to cover the input domain of a given problem, ML methods can make fairly accurate predictions for both interpolations and extrapolations. With the increase of sensor networks, it becomes easier to follow the evolution of a given phenomenon over time and space. This means that the observed data reflects the underlying physical principles that govern its generation. However, here we need to keep in mind the problem of getting reliable data due to the expenses of collecting it. In our case of predicting future state vectors, this type of bias refers to the reliance on historical data for training a model.
- **Inductive bias** - physical equations that are known of governing the system can be

introduced. However, this can only be done for systems that are characterized by relatively simple or well-defined physical principles, so most of the time very complex and laborious implementations are required. For example, to solve differential equations using a neural network, the network can be made to exactly satisfy the respective initial conditions. To illustrate this, considering the previous example of predicting future positions of an object in space using a ML model. If we know that the motion of that object is governed by Kepler's laws, we can incorporate these equations into the model. By encoding the equations into the network architecture and training process, we can ensure that the predictions made by the network adhere to the underlying physics. This means that the network can be designed to exactly satisfy the respective initial conditions or following exactly the known governing equations. By introducing this inductive bias, the network can leverage the known physical principles to make more accurate and physically meaningful predictions.

- **Learning Bias** - Instead of designing a specialized architecture that implicitly enforces prior knowledge (laws and constraints imposed by inductive bias), these constraints are softly imposed, penalizing the loss function appropriately. This approach can be seen as a specific case of multitask learning because an algorithm is simultaneously constrained to fit the data but also to give predictions that approximately satisfy physical constraints. The flexibility of soft constraints allows for the incorporation of more domain-specific knowledge that is harder to be 100% respected. The solutions obtained via optimization with soft constraints can be seen as the equivalent to the *maximum a posteriori* estimation of a Bayesian formulation. For instance, having the aforementioned loss function defined, one can distinctly penalize the part corresponding to how much the data respects the physical equations so that it is possible to find the perfect balance between the data-fidelity part and how much the data respects the governing physical equations.

One way to put physically-informed machine learning to use is by using **PINNs**. These aggregate the information from the observed data with PDEs, incorporating the PDEs into the loss function of the network which can be of many types (high order, integro-differential equations, fractional PDEs, stochastic PDEs). For the architecture of the network, on the left side of Figure 3.1 (NN without being physically-informed) lies the solution of the PDE $u(x,t)$ found by a "traditional" ML approach, and on the right side (physically-informed) represents the residual of the PDE [31]. The loss function can be represented by a weighted sum of both the supervised loss of the observed data of u as well as the unsupervised loss of the PDE resulting in

$$\mathscr{L} = w_{data}\mathscr{L}_{data} + w_{PDE}\mathscr{L}_{PDE}$$

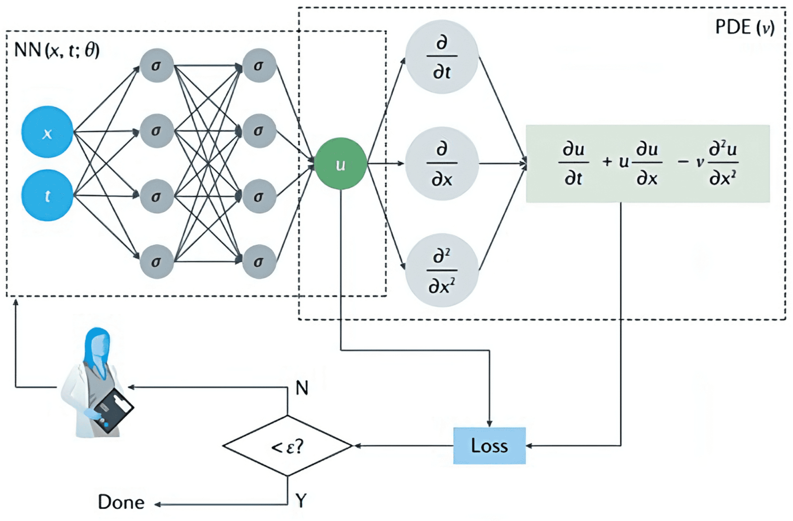

NN (x, t; θ)
x
t
σ
u
PDE (ν)
$\frac{\partial}{\partial t}$
$\frac{\partial}{\partial x}$
$\frac{\partial^2}{\partial x^2}$
$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} - \nu\frac{\partial^2 u}{\partial x^2}$
Loss
< ε?
N
Y
Done

Figure 3.1: PINNs diagram (Burger's equation example) - NN side corresponds to the supervised predictions and PDE correspond to the residual loss. The final Loss value corresponds to a sum of the data loss with the residual loss. The model is trained until the loss value is smaller than a certain defined threshold (ξ).

where the weight values are empirically assigned depending on how much importance we want to give to the predictions made strictly with the data and how much will they respect the physical equations. $\mathcal{L}_{data}$ corresponds to the data loss which is represented by the MSE between the predicted values and the respective labels. $\mathcal{L}_{PDE}$ corresponds to the residual loss which is the difference between the predicted values and the known equations that they need to respect.

Deep learning typically requires large amounts of data for training [5] and, as discussed, in many real-world problems it is very difficult to obtain the desired amount of data. Looking at our problem, acquiring valuable space data means that there is a need to have multiple satellites in orbit logging their positions and velocities continuously and accurately which might not always be the case. In these situations, physically-informed learning has the advantage of achieving strong generalization when less data is available. By enforcing physical equations, these deep learning models are constrained to a low-dimensional manifold thus being able to be trained on fewer data.

For physically-informed learning models there are at least 3 sources of uncertainty. Firstly due to **physics** which represents stochastic physics systems. This is represented by the parametric uncertainty due to the randomness of the parameters (Generative adversarial networks (GANs) are very good for solving high-dimensional stochastic PDEs). Secondly, due to **data**. This uncertainty can appear either due to the presence of noise in the data or because of limited data (e.g. gappy data) which is usually the case in this context of gathering data from satellites. This type of uncertainty can be well addressed with the Bayesian framework that has uncertainty quantification associated with it. However, how to define the prior for Bayesian Physically-informed Neural Networks (bPINNs) in a systematic way is still an open question. Thirdly, due to **learning models**. For example, training and testing errors of NNs are difficult to quantify or the data used could have been handpicked and does not represent reality.

3.1 State of the art

3.1.1 PINNs

Using PINNs might seem like it will give out interesting results immediately because of combining the power ML has due to its non-linear perspective of the data together along with leveraging the knowledge of the physics of the system. However, some problems might affect its usage which some researchers have tried to tackle using various techniques.

In [34] researchers noticed that PINNs tend to fail in slightly complex problems (which can be for example when the PDE coefficients are too large). Problems with a PDE constraint can be thought of as

$$\mathcal{F}(u(x,t)) = 0, \quad x \in \Omega \subset \mathbb{R}^d, \quad t \in [0,T]$$

where $\mathcal{F}$ corresponds to a differential operator representing a PDE, $u(x,t)$ corresponds to the state variable, T corresponds to the last timestep and Ω to the spatial domain. One approach to incorporate physical knowledge is to apply the previous equation as a hard constraint during the training of a neural network which corresponds to the following optimization problem

$$\begin{aligned} \min_{\theta} \quad & \mathcal{L}(u,\theta) \\ \text{s.t.} \quad & \mathcal{F}(u) = 0 \end{aligned} \tag{3.1}$$

where $\mathcal{L}(u)$ is the loss purely from the data and the constraint corresponds to the residual of the PDE that describes that system being 0. In the real world, it is impossible to derive a solution for (3.1), and treating it as a hard constraint can make the problem impossible to solve. By altering the optimization problem, looking at it from another perspective

$$\min_{\theta} \quad \mathcal{L}(u,\theta) + \lambda_{\mathcal{F}}\mathcal{F}(u) \tag{3.2}$$

here $\lambda_{\mathcal{F}}$ works as a regularization parameter to penalize the PDE residual we want to minimize but still allow it to be different from 0. Afterward, typical ML optimization methods are used to minimize the previous loss (3.2). By tuning these parameters, what was observed was that when increasing it (bigger penalization on large residuals) the predictions were better but made the problem much harder to optimize whereas reducing the regularization factor made the problem easier to solve with the disadvantage of coming with a larger error.

To solve this problem, two methods were proposed, curriculum regularization and addressing the problem as a sequence-to-sequence learning task. **Curriculum regularization** works as follows, instead of starting with PDEs with high coefficients from the start, the network starts by training with small coefficients which are easier to learn, and increases them as the training goes on. This helps to "warm start" the network by first finding good initialization weights and progressively making the PDE harder to solve. Looking at the results obtained by the researchers, the error was reduced by at least 1 order of magnitude by applying this technique, and the variance of the error also decreased, thus making the model more stable. **Sequence-to-sequence** focuses on predicting the solution only for the next timestep (t) rather than for all of the time-space. Additionally, when it has to predict the solution for the timestep $t+1$, it uses the predicted value at t as the initial condition.

As it was stated by the investigators of the previous paper, the main cause for the PINNs failure were optimization problems associated with the soft approach concerning the PDE residual inclusion in the minimization problem [34]. More work has been put into this field, specifically in [39]. Since a typical loss function of a PINN can have multiple terms, which can be seen as tasks, it is described as a multi-objective problem

$$\min_{\theta^{sh},\theta^k} \quad \mathcal{L}_k(\theta^{sh},\theta^k) \qquad k = 1,\dots,K$$

where the number of terms is given by K, θ^{sh} corresponds to the shared parameters between tasks, and θ^k to task specific parameters. To simplify the problem what can be done is calculate a weighted (λ_k) sum of the losses of each task ($\mathcal{L}_k$) turning it into a single-objective problem as

$$\min_{\theta^{sh},\theta^k} \quad \sum_{k=1}^{K} \lambda_k \mathcal{L}_k(\theta^{sh},\theta^k) \tag{3.3}$$

It is here where the problem appears, discovering which weights to use for each of the optimization tasks in (3.3) is extremely hard because these individual objectives can have conflicting goals due to countless reasons such as numeric imbalances between variables, the standard deviation not being constant along the training time or by a phenomenon called **catastrophic forgetting** which corresponds to the network forgetting what was learned during the non-physically-informed step before incorporating the domain knowledge with the PDEs.

When dealing with a problem of this sort, some objectives will have larger gradients than others which will bias the optimization process to focus on those objectives and disregard the objectives with smaller gradients, which in the long run will amplify this problem until the tasks with smaller gradients are completely discarded, which is called "vanishing task-specific gradients". This extols the necessity of having a well-defined methodology for discovering which weights can be used to balance each of the terms.

The method proposed that best solves this problem is **Inverse Dirichlet Weighting** which is based on defining the weights based on the variance of the gradient. The aim with this approach is for the weights to have their variances over the back-propagated gradients ($\nabla_{\delta}\mathcal{L}_k$) become equal across all objectives meaning the problem of vanishing task-specific gradients will not be present. This means the weights would be calculated as

$$\hat{\lambda}_k(\tau) = \frac{max_{t=1,...,K}(std\{\nabla_{\theta^{sh}}\mathcal{L}_k(\tau)\})}{std\{\nabla_{\theta^{sh}}\mathcal{L}_k(\tau)\}}$$

where the standard deviation is empirically determined over the vector components. For optimizers that are invariant to diagonal scaling of the gradients (which ADAM is for example [33]), there is a faster way to calculate the weights

$$std\{\nabla_{\theta^{sh}}\mathcal{L}_k(\tau)\} \propto \sqrt{\int_{\Omega_\theta} |\nabla_{\theta^{sh}}\mathcal{L}_k(\tau)|^2 d\theta} \tag{3.4}$$

which means the standard deviation that needs to be calculated to determine the weights is proportional to the inverse of the square root of the Dirichlet energy of each objective (being much more efficient to calculate). This strategy, (3.4), uses weights that are inversely proportional to training uncertainty (defined by the variance of the loss gradients) instead of the uncertainty from the observational noise of the model which other techniques that aim to do the same do (Weighting based on mean gradient statistics [39]).

It is important to state that PINNs are known to be difficult to optimize and converge to an optimal solution as researched in [34, 72, 68, 70]. The application of PINNs encounters challenges in achieving stable training and making accurate predictions. This is particularly notable when the underlying solutions of PDEs involve high-frequency or multi-scale features [21]. Recent research conducted in [71] has attributed this concerning behavior to multi-scale interactions among distinct terms within the PINNs loss function. These interactions engender stiffness in the dynamics of gradient flow, thus imposing stringent stability requisites on the learning rate.

In an effort to alleviate this pathological behavior, in [71] an empirical learning-rate annealing scheme was developed. This method used back-propagated gradients statistics to adaptively allocate weights to the terms in the PINNs loss function. This strategic approach aims to balance the magnitudes of back-propagated gradients. While the implementation of this scheme has been demonstrated to yield noteworthy enhancements in the training and accuracy of PINNs, the reasons behind the complexities associated with training fully connected PINNs remains unclear [21].

3.1.2 Predicting atmospheric density

Currently, the state of the art concerning the modeling of atmospheric density is based on ML techniques along with strong domain knowledge [56].

As stated in [67] the major source of error in predictive models for orbits in LEO is atmospheric drag and the presence of these errors will hinder the predicted satellite positions. The Air Force Space Battlelab's High Accuracy Satellite Drag Model (HASDM) estimates and predicts a dynamically varying global density field.

This new ML-HASDM thermospheric neutral mass density model [56] is based on the database with data from over two decades generated by the HASDM (2000-2020). It works by first applying Principal Component Analysis to reduce the dimensionality of the problem, and afterward applying a traditional machine learning regression model to predict the variable. For this ML problem, there were three loss functions tested, mean squared error, the negative logarithm of predictive density, and continuous ranked probability score. There are also used three different input sets to reduce the bias that may be present using only one. The models generated by this process also take advantage of Monte Carlo dropout to generate probabilistic outputs from which the model uncertainty is calculated.

One of the biggest obstacles for these kinds of physical models is the scarcity of real data measurements [43], particularly in LEO. This region is influenced by the external factors aforementioned (e.g. solar emissions and explosions) that by being measured help the performance of these types of models. These external factors hinder the performance of modeling atmospheric density due to the changes they cause in nominal orbits and since none of them are completely predictable, operators have to make decisions concerning collision avoidance based on uncertainty measurements associated with the predictions since they can largely affect the predicted value [9].

The data used by these investigators contains archived atmospheric density values in LEO for multiple altitudes, latitudes, and longitudes given by HASDM. They also use eight other variables for the solar activity alone, one of them being F10.7 (explained in Chapter 4) and an extra two corresponding to geomagnetic activity. Concerning their implementation, firstly PCA has to be used to reduce the problems of high dimensionality that would make the model impossible to run. Afterward, the data was fed to a feed-forward NN. For the hyperparameters of the network, KerasTuner was used to identify the best combination for every hyperparameter.

For assessing uncertainty quantification there is one ML technique that can be used for this purpose: dropout. Dropout works as a regularization method to prevent overfitting [64] and it uses a binomial distribution with one trial for each neuron with a certain probability P representing if it will be "on" or "off". This means that it reduces the dependence between neurons as the network is forced to learn multiple representations of the same data (by using the different neurons that will have different inputs). Normally, dropout is only used for training purposes, meaning that during the test phase, all of the neurons will be "on". What is applied here is the Monte Carlo dropout which works by applying dropout both on training and testing. This means that the final predictions will not be deterministic and depend on which neurons were randomly chosen to be "on". The goal is to generate random predictions which can be looked at as samples from a probabilistic distribution from which the uncertainty quantification is taken given all of the predictions.

Concerning their results, it was referred that the added historical geomagnetic indices to the dataset improved the performance of the model, reducing the error across the different sets from 0.72% to 2.09% from the errors measured without the added variables. Incorporating the custom loss function NLPD [56] also reduce the calibration error (which is used to judge the reliability of the uncertainty estimates) by an order of magnitude compared to the mean squared error loss.

This work was of particular difficulty during times when solar activity was higher due to additional storms that can occur and not be easily predicted which raises the need for extra data for periods where the AP index (explained in section 4.5) is higher. It was also stated that the dimensionality reduction technique used (PCA) might not be the best because it is based on a linear transformation and when applied to reduce the complexity of the problem, it does not accurately represent the data. Perhaps a non-linear transformation technique would improve the predictions of the model due to a better lower-dimension representation.

As it is possible to notice, even though this model is ML-based, it is rather limited by the need for high-fidelity data which is extremely hard to get in a large quantity as well as the necessity of reducing the dimension of the problem due to the extensive number of variables it has. Since the ML part of this problem is not physically-informed, lacking the domain awareness that physically-informed ML brings to the table by leveraging known physical equations, there is a need for more data which is hard to get. By using a physically-informed neural network instead of a traditional feed-forward network, it is possible to incorporate this knowledge into the predictions made.

Another approach to improve the existing atmospheric density models has been developed in [46]. This work seemed interesting at first sight due to also using Sparse identification of nonlinear dynamical systems (SINDy) to discover the governing equations of a system. This approach focused on using an autoencoder-based model of the thermosphere to obtain a reduced-order representation of the density field in which the dynamics used for this reduced embedding (latent space) would be discovered with SINDy. The pipeline of this work focuses on using previously existing density field data and using it to feed the autoencoder. At this step, SINDy is used to discover the equations in the latent space and together with the known physical equations in LEO, they are used to predict the states from GPS data along with a *Kalman* Filter. At the end of the pipeline, the estimated states for the reduced embedding have to be decoded to retrieve the density field to be compared with the true density. Even if this work is closer to what we will do, it does not follow a physically-informed ML approach, it uses the SINDy equations along with a *Kalman* filter for predicting the states which can have some problems when applied concerning numerical accuracy [63].

3.1.3 SINDy

The approach followed in this work to predict the atmospheric density by following the pipeline of first discovering the equations that rule the dynamical system that describes the orbits along with its perturbations, and integrating them in a PINN afterward, has not yet been done in any research work. With this approach, there are a couple of problems that are solved implicitly because if we can learn equations that describe an orbit, we end up modeling extra unknown parameters for example the attitude of the satellite as well as parameters related to drag by encapsulating everything in a known equation. The state of the art for discovering atmospheric density does not work with as few data as we will be working with, both in measurements and variables, and is not able to do what can be done by the PDEs to describe the orbits themselves but only the mass density parameter itself.

To discover the equations that rule the physical system we will be taking advantage of a state of the art technique, which is **SINDy**. SINDy, proposed by Steven Brunton [65], aims to extract governing equations from observed data while prioritizing sparsity. This trend of employing and using sparsity in dynamical systems is recent but growing in popularity [52, 58]. The fundamental assumption underlying SINDy is that the discovered equations will consist of only a few terms. This sparsity assumption enhances the robustness of the model by reducing sensitivity to noise and preventing the identification of extra residual terms solely due to noisy input. Being insensitive to noise is critical when it comes to an algorithm to identify dynamics from data [59, 13] This goes hand in hand with a parsimonious approach reflecting that a model can be represented by a few terms and does not need extra useless terms that would promote overfitting. This approach introduces a trade-off between complex and sparse models. On the one hand, a complex model accurately captures the intricacies of a system, but it risks overfitting the specific dataset used for model discovery. On the other hand, a sparse model is less complex, incorporates fewer terms, and avoids overfitting. However, it may sacrifice some accuracy compared to the more complex model. In this context, model complexity refers to the number of terms in the discovered equations. We will use a framework developed in Python to leverage the capabilities of SINDy [29].

Considering a set of measurements $x(t) \in \mathbb{R}^n$ at different points in time t, SINDy models the time evolution of such measurements in terms of a nonlinear function f. Thus, the dynamical system for $x(t)$ is given by

$$\frac{d}{dt}x(t) = f(x(t)), \tag{3.5}$$

where $x(t) = [x_1(t), x_2(t), \cdots, x_n(t)]^T$ represents the state of the physical system at time t, and $f(x(t))$ constrains how the system evolves over time.

The implementation of SINDy requires a dataset comprising measurements collected at specific time instances, $t_1, t_2, \cdots, t_n$. Furthermore, the corresponding time derivatives of such measurements are also needed. These datasets are then organized into two matrices:

X, containing the measurements, and $\dot{X}$, which stores the corresponding time derivatives. The user also provides a library of candidate functions, $\Theta(X)$. Such library consists of a set of basis functions that will be applied to the data. For example, the polynomial library used in Chapter 5 would be defined as

$$\Theta(X) = \begin{bmatrix} | & | & | & | & | & \\ 1 & X & X^{P_2} & X^{P_3} & X^{P_4} & \dots \\ | & | & | & | & | & \end{bmatrix}$$

where the polynomials have to be described as, for example for the second degree

$$X^{P_2} = \begin{bmatrix} x_1^2(t_1) & x_1(t_1)x_2(t_1) & \dots & x_2^2(t_1) & \dots & x_n^2(t_1) \\ x_1^2(t_2) & x_1(t_2)x_2(t_2) & \dots & x_2^2(t_2) & \dots & x_n^2(t_2) \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ x_1^2(t_m) & x_1(t_m)x_2(t_m) & \dots & x_2^2(t_m) & \dots & x_n^2(t_m) \end{bmatrix}$$

We want to find a set of sparse coefficient vectors

$$\Xi(X) = \begin{bmatrix} | & | & & | \\ \xi_1 & \xi_2 & \dots & \xi_n \\ | & | & & | \end{bmatrix},$$

where ξ_i defines the coefficients for a linear combination of the basis functions from $\Theta(X)$. Thus, the approximation problem underlying SINDy can be defined as

$$\dot{X} = \Theta(X)\Xi. \tag{3.6}$$

Each column of Ξ corresponds to a sparse vector of coefficients that determine which terms will be active in the PDEs discovered for one of the row equations $\dot{x}_k = f_k(x)$. After the coefficient vectors are determined, each row corresponds to $\dot{x}_k = f_k(x) = \Theta(x^T)\xi_k$.

In practical scenarios, it is common for the data matrices X and $\dot{X}$ to be affected by noise, resulting in deviations from the nominal identity in (3.6). In cases where the measurements in X are relatively clean but the derivatives in $\dot{X}$ are noisy, the equation can be modified to account for this noise

$$\dot{X} = \Theta(X)\Xi + \eta Z, \tag{3.7}$$

where Z corresponds to a matrix of i.i.d. Gaussian random variables with mean zero and standard deviation η.

To solve (3.7) for Ξ to find the coefficients, there are a couple of approaches that can be taken from using LASSO [23] which promotes sparsity but can be computationally expensive for large datasets or an algorithm like sequential thresholded least-squares [65].

With this algorithm, it starts by finding a least-squares solution for the variable Ξ. Subsequently, coefficients that fall below a predetermined threshold value λ are truncated. Once the indices of the non-zero coefficients are identified, another least-squares solution targeted towards these indices for Ξ is calculated. The newly obtained coefficients also undergo the cutoff using λ, and this is repeated until the non-zero coefficients converge to a sparse solution. Notably, this also has the advantage of only needing one parameter, λ to dictate the extent of sparsity in Ξ.

SINDy needs the correct function library in order to correctly identify the dynamics which can be hard to know [65]. Here, physics knowledge can help leverage the usefulness of data to steer the process of choosing the correct functions to simplify dynamics.

In recent years, there has been growing interest in applying the SINDy algorithm to extract governing equations and uncover hidden dynamics from observational data. While SINDy has shown promise in various applications, such as discovering equations of motion and identifying relevant terms in dynamical systems, its specific utilization for predicting state vectors of satellites has received limited attention in the literature.

Several studies have successfully applied SINDy to various domains, such as fluid dynamics [20] or biochemical systems [40]. Moreover, SINDy has proven effective in identifying the equations of motion and understanding system behavior in mechanical systems. For example, in [27], the authors employed SINDy to reveal the underlying mathematical model governing the motion of a damped double pendulum. The authors successfully captured the system dynamics and accurately predicted its motion using the derived equations by analyzing experimental data. While SINDy has not been applied explicitly for predicting state vectors of satellites, it has indeed found applications within space research. One such notable example is [47], where SINDy was employed to derive best-fitting differential equations governing the spatial and temporal evolution of the thermospheric density field. This approach allowed for real-time density estimation, an essential factor in understanding the dynamics of space objects in LEO due to atmospheric drag. In [47], the ability of SINDy to extract governing equations from observed data while promoting sparsity was demonstrated in the space domain, enabling a concise and interpretable representation of the thermospheric density dynamics. The effectiveness of the method in this context showcases its potential for understanding complex systems within space research.

The rapid expansion of global satellite communication companies, advancements in miniaturized satellites, and revolutionary ideas such as autonomous nanosatellite swarms [25] have significantly amplified the potential for conflicts and collisions among these orbiting entities. Consequently, ensuring accurate and timely trajectory predictions for space objects has become crucial to establish a solid foundation for present and future space situational awareness systems. The traditional physics-based models used for orbit prediction often fail to achieve the required accuracy, leading to collisions due to the lack of essential information about the space environment and characteristics of the Resident Space Objects (RSOs), which can be challenging to acquire. Machine Learning techniques have been used to predict satellite state vectors. In [50], the authors address the challenges of efficiently and accurately predicting the orbit of RSOs for space situational awareness and collision avoidance purposes. The growing population of space objects in orbits in LEO has recently become a primary concern for space situational awareness [17]. To overcome these limitations, the authors in [50] hypothesize that a machine learning approach can learn the underlying patterns of orbit prediction errors from historical data. They specifically explore using Support Vector Machines (SVMs) [69] to enhance the accuracy of orbit predictions. The SVM model is designed and trained at a current epoch and then utilized to reduce the orbit prediction error at a future epoch. Through simulations involving RSOs in a Sun-synchronous Orbit (SSO), the study demonstrates that the trained SVM model effectively captures the underlying relationships between the learning variables and provides desirable predictions for the orbital motion. It shows promising results with good average and individual performance in reducing prediction errors. The paper also indicates that there is a limit to the improvements once sufficient data has been utilized for training the model. One drawback is that it needs to be updated frequently in practice. Orbit predictions should not be made too far into the future. The findings of this paper add to the body of research exploring machine learning approaches for orbit prediction accuracy improvement. While our work focuses on applying SINDy to discover the governing equations of satellite motion, using an SVM in orbit prediction showcases the potential of various machine learning techniques in enhancing space-related predictions. The combination of diverse approaches can contribute to advancing space situational awareness and managing space objects in the future. However, despite the wide-ranging applications of SINDy, its direct application for predicting state vectors of satellites from observational data remains unexplored. To the best of our knowledge, no previous work has specifically investigated the use of SINDy or similar methods to directly predict state vectors of satellites, maintaining the physical meaning of the variables predicted. This represents a significant gap in the literature, as an accurate and interpretable prediction of satellite trajectories is crucial for multiple space-related applications, including orbit determination, collision avoidance, and mission planning. Through this research, we aim to demonstrate the effectiveness of SINDy for predicting state vectors and contribute to the broader field of satellite trajectory analysis. By leveraging the vast amount of available observational data in conjunction with high-fidelity simulators, we strive to enhance our understanding of the dynamics and interactions that drive satellite motion, thus enabling improved satellite operations.

4

Exploratory Data Analysis

Before changing the theme of this work from a theoretical to a more practical perspective it is important to understand the data that we will work with. This chapter will be dedicated to Exploratory Data Analysis on the dataset used hitherto which has the probability of being changed due to reasons that will be explained in this section.

4.1 Dataset

The dataset used is publicly available from Planet [36] and by default when accessed, it downloads a file that corresponds to data from that day. Before creating a script to download data from the beginning, we first analyzed that single file to understand what we would work with. This first file corresponded to 28-07-2022 and had 338 samples and 10 columns: satelliteID, epochSince, positionX, positionY, positionZ, velocityX, velocityY, velocityZ, ballistic drag coefficient, and SRP as seen in 4.1.

All columns will be used apart from SRP (which represents solar radiation pressure) for not being currently fit to the data. Each sample in this context can also be referred to as a state vector. A state vector corresponds to the state of a satellite. It consists of the position and velocity values in each axis at a given epoch.

After understanding the data we had in hand, we quickly realized that it would be better to download every file so that we could detect and analyze the evolution of the variables over time. The state vectors files downloaded and used were all given by Planet from 15-04-2020 up until 28-12-2021.

For this analysis, we decided to convert the state vectors from a cartesian coordinate system to a keplerian orbit representation. The reason behind this choice was that it is not intuitive, for the analysis part at least, for a human to look at an orbit defined by these variables. For example, if one wants to see how much an orbit plane rotates over time, it is not directly interpretable with cartesian coordinates on how that would be seen and proved. However, with a keplerian orbit, it is possible to see the right ascension of the ascending node changing over time which describes just that.

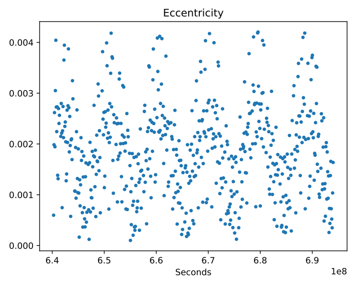
Name	**Units**	**Description**
Satellite ID	-	Satellite Hardware ID (HWID, a 4-digit hex number)
EpochSince	Seconds	Seconds since J2000 epoch Terrestrial Time
PositionX	Meters	X-coordinate position of the satellite in J2000 frame
PositionY	Meters	Y-coordinate position of the satellite in J2000 frame
PositionZ	Meters	Z-coordinate position of the satellite in J2000 frame
VelocityX	Meters/Second	X-coordinate velocity of the satellite in J2000 frame
VelocityY	Meters/Second	Y-coordinate velocity of the satellite in J2000 frame
VelocityZ	Meters/Second	Z-coordinate velocity of the satellite in J2000 frame
Ballistic Drag Coefficient	Kilogram/$Meters^2$	Coefficient representing atmospheric drag effects
SRP ballistic coefficient	Kilogram/$Meters^2$	Solar Radiation Pressure acting on the satellite

Table 4.1: Variable Descriptions

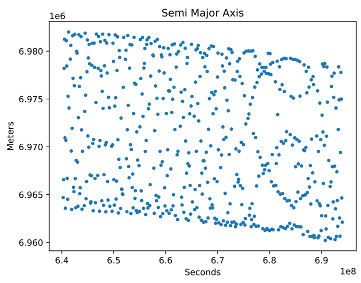
4.2 Variations over time - orbital elements

To have an idea of how these variables change over time, a visual representation is the best way of getting a glimpse of what is happening which passes the responsibility of understanding the patterns to us. Referring back to chapter 2, where the orbital elements of a keplerian orbit were explained, we will now use all of the measurements over the time available concerning the satellite with ID 0903 to look at how these variables varied, shown in Figure 4.1. Firstly, the argument of perigee looks like having a pattern repeating itself over time with its value ranging from 4 radians to -6 radians and a period of around 115 days. Concerning Right Ascension of the Ascending Node (RAAN), mentioned in the previous paragraph, it is possible to see it varies continuously linearly which means that the orbit plane rotates completely back to its original orientation after around 1 year.

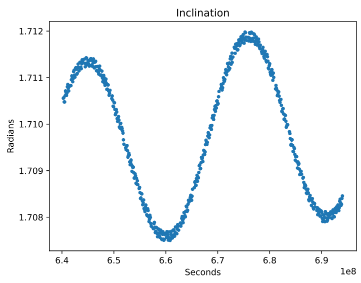
The eccentricity of the orbit made by this satellite appears to be varying around values near 0, which means it is an almost circular orbit. It also looks like it is periodical, with its value around 115 days which coincides with the period of the argument of perigee. Relatively to the semi-major axis, this variable looks like it has a rather uniform distribution with a slightly decreasing tendency. This makes sense because, in the orbit a satellite does, its height related to the surface of the Earth also reduces slightly due to the effect of atmospheric drag. Concerning the inclination of the orbit, this variable is well described with a sinusoidal-looking wave, varying the angle made between the orbital plane and the equatorial plane from around 97.86^{o} to 98.09^{o} which is a really small disturbance



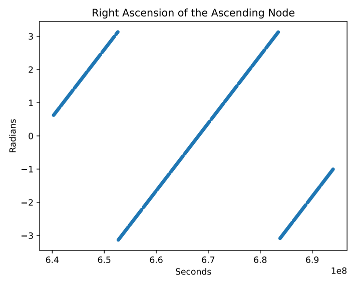
(a) Eccentricity



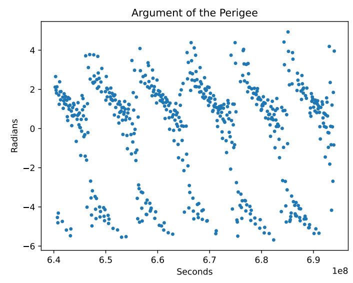
(b) Semi-major Axis



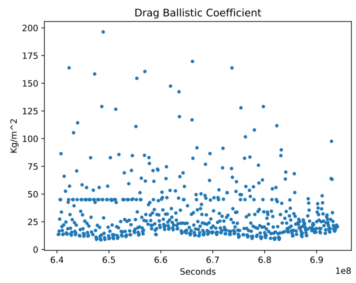
(c) Inclination



(d) RAAN



(e) Argument of the Perigee



(f) Drag Ballistic Coefficient

Figure 4.1: Variables for satellite ID 0903 - It is possible to see the periodicity in all of these variables (except for the drag ballistic coefficient) which show that a specific pattern for most of them is repeated over time.

but it shows that it is constantly "wobbling" between those two values. Looking at the drag values, it is possible to see that the majority is concentrated around 20 kg/m^2 with a couple of outliers reaching 200 kg/m^2. These high drag values might be strange to think about considering the satellites used on this dataset by Planet are rather small, having an approximate parallelepiped shape with dimensions of around 10x10x30 cm. However, since the attitude of the satellite is not available, the surface area considered changes drastically depending on where it is pointed to. For example, the surface ratio between being with the solar panels open in the movement direction compared to being sideways can be at least 8 times more, making drag reach those high values.

4.3 Variable Distributions

After verifying the evolution of the variables over time, we checked if any of them followed a known analytic distribution. This is useful because a distribution of this form is characterized by a mathematical function [15], the cumulative distribution function (CDF), which represents a simple way of describing how the data is distributed, leaving out unneeded details. The CDF of a random variable X is described by $F_X(x) = P(X \leq x)$ which represents the probability of X having a smaller or equal value to x. Firstly, to check if a variable follows an **exponential distribution**, one can do a statistical test to verify it. One of the most used for checking if one is in the presence of an exponential distribution is the *Lillerfors* test [38]. The null hypothesis considers the data comes from an exponential distribution. If the null hypothesis is rejected, it is not possible to verify that a variable follows an exponential distribution.

After repeating this test for all of the variables, none of them followed an exponential distribution since the null hypothesis was rejected for all of them. What was done was to see if they followed it approximately. This can be done by plotting the complementary CDF with the y-axis in a logarithmic scale and checking if the result is a straight line. This works because, since the CDF of an exponential distribution is given by $CDF(x) = 1 - e^{-\lambda x}$ which means its complementary is

$$y = 1 - CDF(x) \Leftrightarrow y = 1 - (1 - e^{-\lambda x}) \Leftrightarrow y = e^{-\lambda x}$$

$$\therefore \log y = -\lambda x$$

which means that on a log-y scale, CCDF is a straight line with its slope equal to $-\lambda$. However, when plotting this graph for all of the variables, none of them follow a straight line which represents that they can not be described with an exponential distribution (I).

The other probability distribution we tried to verify if any variables followed it, was with the normal distribution which is the most commonly used, mainly due to its simplicity and ability to model many different phenomena. There were two ways we did this. Firstly we tried the more formal *shapiro-wilk* normality tests on the variables but unfortunately, none of them proved to be normally distributed. Secondly, to just have a more informal look at "how normal" the data are, we decided to use a normal probability plot which on the y-axis has the variable to be tested, and on the x-axis, the standard deviations from the mean. Having the perfect model line as grey and the data line as blue, if both of them match between two values of standard deviation, it is possible to say that they are approximately normal within those standard deviations. However, when testing it for all the variables, some respected the normality up to $3 * standardDeviations$ from the mean but none of them can be said to be normally distributed (II).

4.4 Correlation

One important analysis to do is to check the relationships between variables which can be done by checking the correlation. One way to do this would be to calculate Pearson's Correlation [60] since it is easy to interpret, if this value is either -1 or 1 the variables are perfectly correlated. The problem is that if the value is 0, even though it is tempting to say that those variables are not correlated, this cannot be concluded due to Pearson's Correlation only measuring linear relationships. The alternative is to simply plot a scatter plot comparing each of the variables with one another, helping us to see if any variables are strongly correlated meaning that one can be dropped, reducing the complexity of the problem we are dealing with.

Analyzing the existing correlations III.1, it looks like almost all of the variables are correlated in some way or another, although the relationships are mostly non-linear. Some

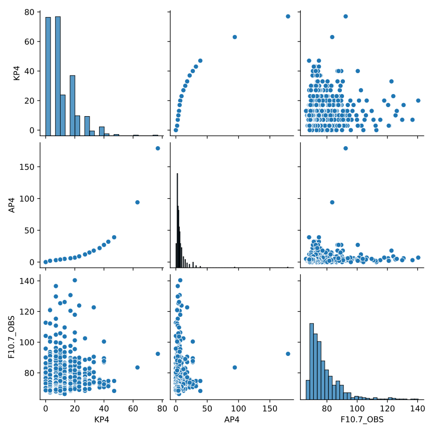


Figure 4.2: Correlation Plot - External variables for satellite ID: 0903. These external variables describe the solar effect and might add more information that explain some phenomena that the orbital elements cannot.

variables were expected to be correlated, for example, the velocity in the Z axis with the position in the same axis but there were some others not as obvious, for example, the ballistic drag coefficient with the semi-major axis.

4.5 Additional External Data

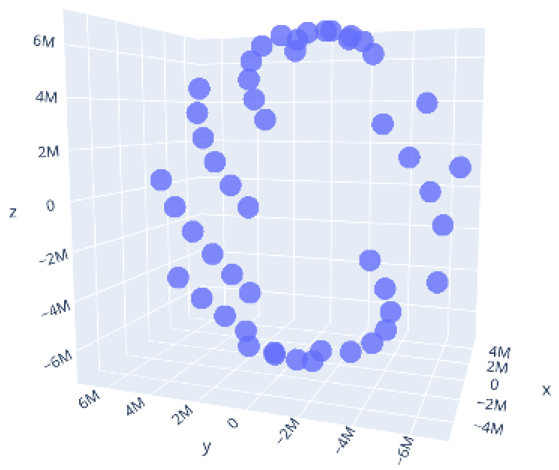
When analyzing the data, since in the previous analysis drag was not strongly correlated with any other variable, we tried to aggregate extra information to the dataset that could help explain either drag or other variables better. The three extra variables introduced were A_p, K_p and $F_{10.7}$.

K_p is used to characterize the magnitude of geomagnetic storms and serves as an indicator of disturbances in the earth's magnetic field. However, K_p is measured in a logarithmic scale and is not linear with the fluctuations of what it is measuring. Because of this non-linear relationship taking the average of multiple K-indices is not meaningful. Due to this, every 3-hour K-value is converted into a linear scale which is the A_p representing an average measure of the global scale of disturbance of the Earth's magnetic field. $F_{10.7}$ is the solar radio flux at 10.7cm (2800 MHz) which is an excellent indicator of solar activity.

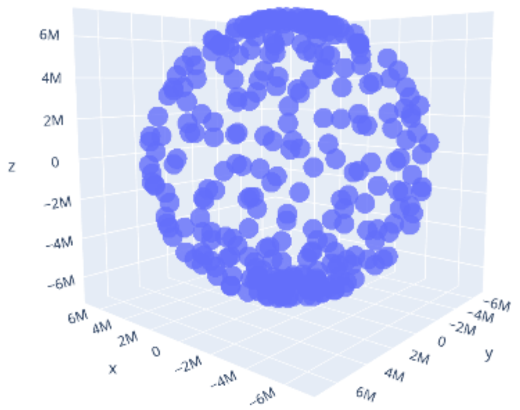
These three new variables introduced that describe the solar perturbance can possibly describe the dataset a bit better and may explain certain outlier values. However, when looking at the correlations between these new variables and the rest of them, there was no visible correlation. There was a relationship between A_p and K_p that looks exponential which matches up with the meaning of the variables since one of them is in a logarithmic scale and the other is in a linear scale. Both of these variables do not seem to be correlated with any other variable. $F_{10.7}$ is not correlated with any variable (Figure 4.2).

4.6 Data Fidelity

One problem referred at the beginning of this chapter was the possibility of having very little data to properly make predictions in the future. This is due to only having a single state vector per satellite per day. Considering satellites in LEO can take only between 90 minutes and 2 hours to complete a full orbit [57], it is easy to understand that having only a single measurement per day is not enough to accurately describe its orbit. Looking at the first 50 state vectors of a single satellite (corresponding to 50 days), it is possible to see its orbit drifting.



(a) Data from 50 days



(b) Data from 2 years

Figure 4.3: Evolution of the position of Satellite 0903 over time

Due to the scarcity of measurements from real-world data per trajectory, we employed a realistic high-fidelity propagator used in Neuraspace that accounts for various exogenous perturbations, including solar radiation pressure, atmosphere density models, and gravity variations due to the oblateness of the Earth. Additionally, we considered internal information specific to each satellite, such as its reference area, drag coefficient, and mass. Utilizing this propagator, we generated a more finely-grained dataset, significantly increasing the number of measurements available for analysis. This data augmentation process enables the application of the SINDy methodology.

The resultant dataset consists of state vectors, each representing the complete state of a satellite. These vectors encompass both the position and velocity values along each axis, providing a comprehensive depiction of the motion of the satellite in three-dimensional space. With this enriched dataset, we can explore the capabilities of SINDy in uncovering the underlying governing equations of the dynamics of the satellite system.

5 Preliminary Work

What is done in this Chapter is in the field of discovering partial differential equations from data using SINDy, which was explained in Chapter 3. Considering that SINDy can find PDEs that explain the data that it was fitted with and PINNs will need PDEs to incorporate the said physics laws in the predictions made. Having this in mind, learning how to take the best advantage of SINDy and analyzing its performance is a crucial step before constructing any kind of PINN.

5.1 Candidate Nonlinear Functions

SINDy requires the appropriate choice of a coordinate system and function basis to capture the sparse dynamics of the system accurately. However, such steps can be challenging and nontrivial [65]. In this context, domain-specific knowledge of the underlying physics can be priceless. By leveraging physics knowledge, one can use the power of data to guide the selection of appropriate coordinates and simplify the dynamical model of the system. This interplay between domain expertise and data-driven analysis facilitates the discovery of meaningful and interpretable system behavior models.

In this work, two different function bases were considered to explore the modeling of the nonlinear dynamics of the system. The first function basis is grounded on the underlying physics of the problem, aiming to capture the intrinsic relationships and principles governing the behavior of the system. The second function basis consists of polynomial functions, which are more general and widely applicable due to the flexibility of polynomials, enabling the exploration of more straightforward and more interpretable representations of the data.

5.1.1 Domain-Driven Custom Functions

We begin by utilizing a custom functions library, with the primary objective of assessing the capability of SINDy to identify the correct terms among the available options for constructing the equations. As mentioned in Chapter 4, the data consists of a time series of state vectors containing the positions and velocities of a given object. Thus, we seek to find the first-order PDEs of the underlying system. For a given state vector,

$$w = \begin{bmatrix} x & y & z & \dot{x} & \dot{y} & \dot{z} \end{bmatrix},$$

to find equations that accurately describe

$$\dot{w} = \begin{bmatrix} \dot{x} & \dot{y} & \dot{z} & \ddot{x} & \ddot{y} & \ddot{z} \end{bmatrix}. \tag{5.1}$$

From the orbital motion equations [12], we can approximate $\dot{w}$ by neglecting the contribution of external forces. Thus, $\dot{w}$ is given by

$$\dot{w} \approx \begin{bmatrix} \dot{x} & \dot{y} & \dot{z} & \gamma x & \gamma y & \gamma z \end{bmatrix} \tag{5.2}$$

where $\gamma = \frac{\mu}{(x^2+y^2+z^2)^{3/2}}$, and μ is the standard gravitational parameter of the Earth. In this sense, the proposed domain-driven custom library contains the terms in (5.2), and the main purpose of this trial is to study how can SINDy recover the well-known parameters for the standard solution of orbital motion. Writing this in vectorial form we get

$$\left[\frac{d^2x}{dt^2}, \frac{d^2y}{dt^2}, \frac{d^2z}{dt^2}\right] = -\frac{\mu}{(x^2+y^2+z^2)^{\frac{3}{2}}}[x, y, z] \tag{5.3}$$

By Equation 5.3, we can build a library made out of the following terms

$$\left\{x, \frac{x}{(x^2+y^2+z^2)^{3/2}}, \frac{y}{(x^2+y^2+z^2)^{3/2}}, \frac{z}{(x^2+y^2+z^2)^{3/2}}\right\}$$

5.1.2 Polynomial Functions

As mentioned, a polynomial library was also considered. The goal of incorporating polynomial functions is to explore a more general and flexible approach to represent the observed state vectors. By considering polynomial terms of varying degrees, we aim to assess the possibility of capturing the nonlinear system dynamics using simpler and more interpretable terms without relying on domain-specific knowledge. In this work, terms up to degree four were considered, which allow for capturing a wide range of nonlinear relationships and interactions within the system. By limiting the degree to four, we aim for a practical balance, enabling us to capture important nonlinearities while maintaining a manageable number of terms in the model.

5.2 Results

To assess the performance of our approach, i.e., the effectiveness of the equations obtained through the application of SINDy, such equations were used to propagate the data over time. The resulting data points were then compared to the observed trajectory of the satellite. This comparative analysis enabled us to evaluate the predictive accuracy of the methodology and its reliability.

5.2.1 Choice of the SINDy Optimizer

One of the key decisions when applying SINDy is the choice of the model optimizer [62]. However, after choosing one, it is still highly dependent on multiple factors revolving around the data fed (if they are standardized or not, if multiple trajectories are used, if the data contain drag or not, and finally if they are the result of custom functions or polynomial terms), the differentiation method chosen and even with everything fixed, there is still some variance on the identified equations comparing multiple runs [62, 28]. Several optimization methods are used in the literature [29].

However, we have decided to proceed with the Forward Regression Orthogonal Least-Squares (FROLS) [54] since it offers advantages in terms of interpretability, computational efficiency, and robustness to noise. FROLS tries to solve the following optimization problem:

$$\min_{v} \quad \|t - Av\|_2^2 + \alpha\|v\|_2^2 + b\|v\|_0 \tag{5.4}$$

where $b = \kappa N$, N is the condition number of the matrix Θ which corresponds to the function library whose columns represent the set of basis functions and $\|u\|_0 = \sum_{n=1}^{N} |u_n|^0$ corresponding to the L0 norm that is the number of non-zero entries in u. This optimizer has two tunable parameters, α and κ. α represents the optional L2 regularization on the weight vector to enforce smaller coefficients and κ is also an optional parameter that if used, computes the mean squared error with an extra L0 regularization term with strength equal to b above-mentioned.

5.2.2 First order PDE

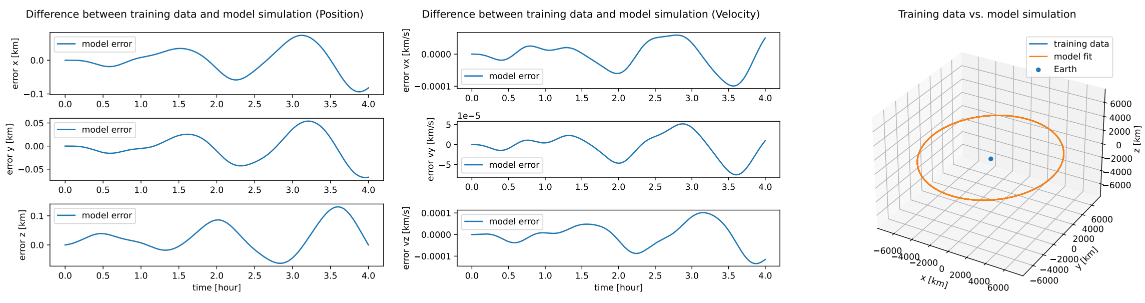
5.2.2.1 Custom Library

The goal with this library was to see if SINDy managed to find the correct terms out of the available ones to reconstruct the correct equations and if it could find a specific coefficient. For numerical stability purposes, we used the data in kilometer units. After applying SINDy to the data obtained from propagating the initial state vector over three hours, the solution, as represented in (5.2), was accurately determined. Notably, the gravitational parameter was correctly identified up to the ninth decimal place, indicating the precision achieved in the estimation process.

$$RealValue = 3.986004418 \times 10^{14} m^3 s^{-2}$$

$$PredictedValue = 3.986004412 \times 10^{14} m^3 s^{-2}$$

When looking at the errors attained for a span of four hours as seen in 5.1, it is possible to see that the errors are low, having mean values for the positions as -0.0022 km, -0.0033 km and 0.025 km for each axis respectively and for the velocities -5.63×10^{-6},-4.66×10^{-6} and -4.38×10^{-6} for each axis. However, one has to keep in mind that here we have no external non-conservative forces acting on the satellite that would make the predictions harder. This experiment has the objective of finding the standard gravitational parameter



(a) Errors in Positions (b) Errors in Velocities (c) Orbit Dynamics

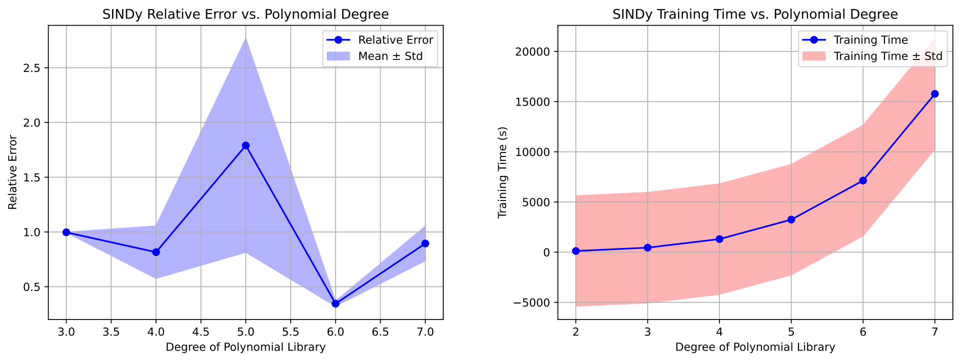
Figure 5.1: Errors obtained when comparing the true values with the ones obtained using the custom library.

of the Earth from data. It is also important to state that we are specifically saying which terms will have to be present in the identified PDEs which eases the difficulty of the predictions by the model.

These are the equations known to describe an orbit for centuries which is interesting in being able to find them from only measurements. However, one question arises. Is it possible to reproduce an orbit with much simpler equations?

5.2.3 Polynomial Library

An orbit can be represented by the custom functions as above-mentioned. However, we are specifically saying what terms have to be present in the identified PDEs. An alternative is to use a library simply made out of polynomial terms up to a certain degree (in this example, up to the fourth degree) and check if the PDEs found can also encode the physical laws that determine an orbit. This has the advantage of being faster to run since polynomials are really simple and nicely fit the data.



(a) Relative Errors (b) Training Times

Figure 5.2: Relative errors across all axes and the correspondent training times over using function libraries up to different degrees

In order to understand the impact of different degrees in polynomial libraries with

SINDy, we analyzed the training times, revealing an interesting observation. As it is seen in Figure 5.2(b), it is possible to see an increase in training times for SINDy when employing polynomial libraries of diverse degrees (ranging from 3 to 7) with an exponential behavior. This trend is attributed to the expanding size of the function library within SINDy which needs to comprehend all of the combinations between the terms.

In addition, a visual representation displays the average relative errors across all axes in Figure 5.2(a), which enables us to analyze the trade-off between the size of the library and its predictive efficacy. By doing so, we decided that using a library up to degree 4 would be ideal choice. This decision strikes a balance between manageable training times and achieving a satisfactory level of relative errors. If we decided to choose a library with terms up to a higher degree that would result in lower training errors, for example up to the 6th degree, we would have a higher training time but more importantly, we would be overfitting to the training data due to the higher polynomial terms.

This analysis emphasizes the importance of correctly defining the problem and using the correct set of tools in order to solve it, in this case being which function library to use. Such decisions will influence the training process duration and predictive accuracy. With these observations in mind, we move forward, building on the lessons gained from this initial exploration.

Using the polynomial library, SINDy could identify the first-order PDEs for the positions correctly but did not manage to find any equations for the PDEs for the velocities.

$$\begin{aligned} \dot{x}_{\text{calculated}} &= c_1\dot{x} & \ddot{x}_{\text{calculated}} &= 0.00 \\ \dot{y}_{\text{calculated}} &= c_2\dot{y} & \ddot{y}_{\text{calculated}} &= 0.00 \\ \dot{z}_{\text{calculated}} &= c_3\dot{z} & \ddot{z}_{\text{calculated}} &= 0.00 \end{aligned}$$

where $c_1 = c_2 = c_3 = 1.00$.

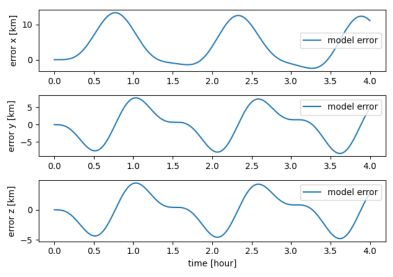
Upon examining the range of values for each variable, we observed significant differences in orders of magnitude. This discrepancy posed a challenge for the optimization step of SINDy, which aims to identify the correct PDEs that best fits the data. To address this issue, we considered standardizing the data to alleviate the impact of varying scales.

By fitting the model with standardized data, we achieved a straightforward polynomial PDEs that accurately described the orbit:

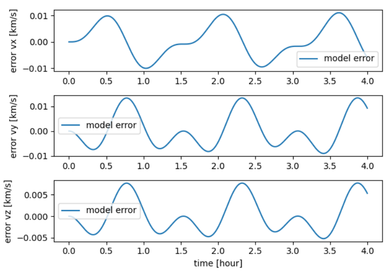
$$\begin{aligned} \dot{x}_{\text{calculated}} &= c_1\dot{x} & \ddot{x}_{\text{calculated}} &= c_4 x \\ \dot{y}_{\text{calculated}} &= c_2\dot{y} & \ddot{y}_{\text{calculated}} &= c_5 y \\ \dot{z}_{\text{calculated}} &= c_3\dot{z} & \ddot{z}_{\text{calculated}} &= c_6 z \end{aligned}$$

where $c_1 = c_2 = c_3 = 0.001$ and $c_4 = c_5 = c_6 = -0.001$.

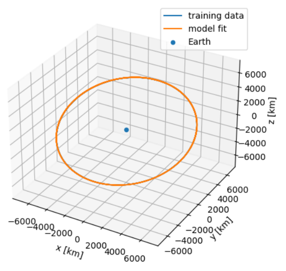
When training the model with one orbit and simulating with it, the resultant errors on the positions reached a magnitude of around 10 km on the worst performing axis (i.e., the x-axis). On the velocities, the worst obtained error was approximately 0.01 kms^{-1} as seen in Figure 5.3.



(a) Errors in Positions



(b) Errors in Velocities



(c) Orbit Dynamics

Figure 5.3: Plots for the errors on the positions and the velocities and the resultant orbit learning with only one trajectory

In our investigation, we also explored the utilization of pre-computed derivatives in conjunction with SINDy. The idea was to supply the model with exact derivatives of the data points as an argument during the fitting process, referred to as using the $\dot{x}$ argument. We pre-calculated the time derivatives based on the known differential equations governing the data, creating a matrix of these derivatives. However, upon careful evaluation, we observed that this approach yielded suboptimal results compared to not using them, as it was done hitherto. Despite the initial appeal of mitigating noise amplification, the simulated orbit generated using pre-computed derivatives did not align with the expected trajectory. As a result, we have decided not to include this alternative approach in our final analysis, as it did not yield desirable outcomes.

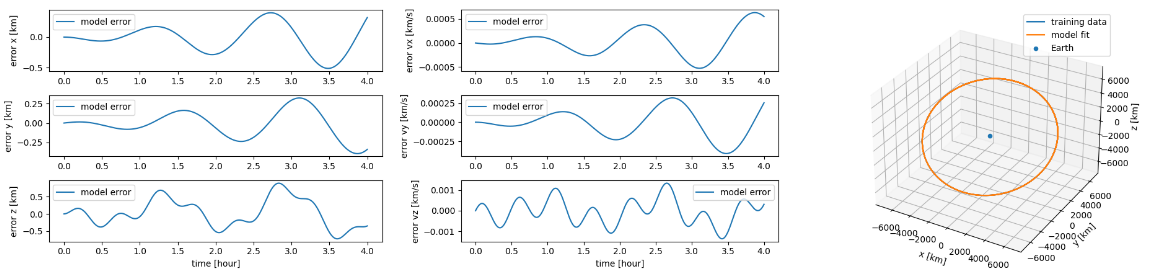
5.2.4 Multiple Trajectories

Another approach tested was learning from multiple satellite trajectories which supposedly would help the model have more data with a more complete spatial distribution. To do this, we selected the data from all the satellites and, for each one (for each ID), we integrated the differential equation 5.3 so that we could get a dataset with no measurement noise (over a span of 3 hours). This resulted in a list with a length equal to the number of different satellites, where each index is a matrix representing the simulated dataset for that satellite. Along with this list of orbits, there is another list with the same length but with the corresponding times for each state vector of each satellite. Using this approach, we were getting worse results than only using one orbit, the errors were increasing, and the resulting orbit did not follow the correct trajectory. We found this **counter-intuitive** since learning with more data gave a worse performance.

To understand why adding more data to train the model resulted in worse predictions than only training with one orbit, we hypothesized that the problem could be that the orbits used were too similar and could be acting as noise rather than extra useful data.

Consequently, instead of using each initial state propagated, the orbits were created by varying their inclinations, eccentricities, and altitudes to get a representative dataset of different trajectories around LEO as stated in 4.6. After generating this dataset and training the model with it, we applied it to see if it could accurately represent it given the initial state of an orbit. We created a different dataset to have more representative orbits with varied orbital elements as a way of us controlling what was being used for training. The orbits created had combinations of inclinations ranging from 0^o to 180^o with 15^o intervals, altitudes from 200 km up to 1200 km with 200 km intervals, and three eccentricity values: 0.01, 0.02, and 0.03.

When simulating the model with the same orbit used for the single orbit test but now training it with the multiple orbits aforementioned, the results improved drastically as seen in figure 5.4. This proves the hypothesis of the extra data not being representative. Using multiple orbits, it was possible to decrease the error on the positions on the worst performing axis to only around 0.5 km and for the velocities as low as 0.001 kms^{-1}. This is informative and proves beyond a reasonable doubt that learning with more representative data is better than with less data.



(a) Errors in Positions (b) Errors in Velocities (c) Orbit Dynamics

Figure 5.4: Plots for the errors on the positions and the velocities and the resultant orbit learning with multiple trajectories

5.3 Data with full dynamics

Constructing a simulated dataset makes it possible to introduce realistic full dynamics related to drag and interactions with other celestial objects and see if SINDy manages to find the extra terms related to the added force. Drag corresponds to an extra term in the equations that define the derivative of the velocities which is subtracted from the acceleration value and is given by

$$\vec{a}_{drag} = \frac{1}{2}\rho C_D \frac{A}{m} \|\vec{v}\|\vec{v}.$$

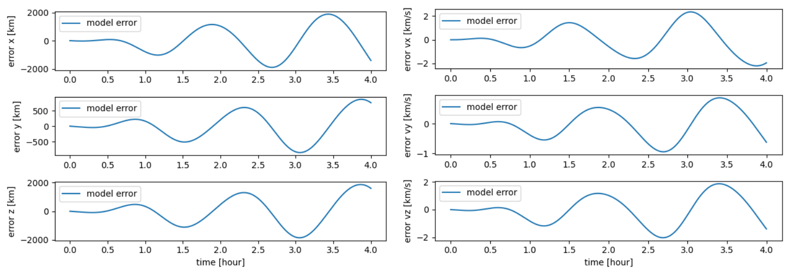
The dataset used corresponds to the same orbits with realistic values for ρ resulting in realistic drag values and a more complex gravity modeling. It can be seen as a high-fidelity dataset with only differences from a real dataset due to noisy measurements.

5.3.1 Custom Library

Firstly, for testing, if it could find the exact terms for the equation that are known to represent drag, three extra functions had to be added to the library corresponding to the drag terms for each axis. Unfortunately, even with the presence of drag, making the satellite fall towards the earth at a rapid pace, the PDEs found, even though having extra terms compared to having no drag, the resulting orbit had the same contour as without drag which means the equations could not capture this effect. When repeating the same process for standardized data, a couple of residual terms appear on the equations, however, the orbit looks the same as without drag.

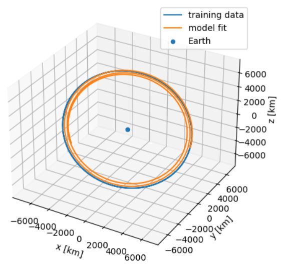
5.3.2 Polynomial Library

Testing if the added polynomial terms could capture enhanced dynamics, repeating the process by training the model with only one orbit and simulating with a different one, the results were obviously worse than simpler dynamics. With standardized data, the orbit found spirals slightly into itself as seen in figure 5.5 which might be an amplification of what really happens due to the decrease in the altitude of the satellite.



(a) Errors in Positions

(b) Errors in Velocities

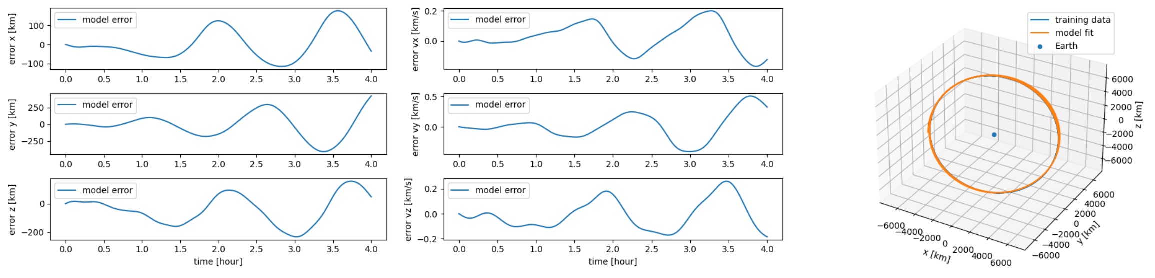


(c) Orbit Dynamics

Figure 5.5: Graphics for the errors on the positions and the velocities and the resultant orbit learning with a single trajectory and full dynamics

5.3.3 Multiple Trajectories

Using standardized data and training the model with the descriptive high-fidelity dataset, the results were much better. Training with multiple trajectories and simulating with a different one with an unseen inclination, eccentricity, and altitude, decreased the error compared to only using a realistic orbit. In this case, it is possible to see the outline of the orbit in Figure 5.6 follows much more precisely the correct trajectory and does not suffer from incorrect spiraling.



(a) Errors in Positions (b) Errors in Velocities (c) Orbit Dynamics

Figure 5.6: Plots for the errors on the positions and the velocities and the resultant orbit learning with multiple trajectories and full dynamics.

5.4 Second order PDEs

Until now what was being done was calculating the first-order PDEs since it is what SINDy was made to do. But is it possible to find the second-order PDEs for the data? By calculating $\dot{x}$ (which was already calculated to be used as an argument for the first-order case) and fitting SINDy with the first three columns of the $\dot{x}$ matrix (corresponding to the first derivatives of the positions, the velocities), we get PDEs for the velocities. Simulating with these equations what we get is a matrix consisting of the velocities simulated along the timesteps desired. Currently, this section is still work-in-progress and might not be used afterward, thus it was not thoroughly experimented with. At this point, to compare if what was done is correct, if we integrate this matrix, this will give us another matrix consisting of the positions. To do this, the main idea is that to calculate a new position in a timestep we have to add the current position to the displacement made over that timestep. To calculate this displacement we have that $\dot{u} \approx \frac{\Delta u}{\Delta t}$, and $\Delta u = \dot{u}\Delta t$

Given that we have the initial position p_0, the matrix can be calculated the following way and repeat the process until all of the velocity matrix has been iterated.

$$p_1 = p_0 + v_0(t_1 - t_0)$$

$$p_2 = p_1 + v_1(t_2 - t_1)$$

5.4.1 Polynomial Functions

Applying what has just been said to the polynomial functions library with the data standardized and integrating the simulated velocities to get the positions resulted in a nicely defined orbit with its equations containing high coefficients

$$\ddot{x} = c_1\dot{x} + c_2\dot{y} + c_3\dot{z}$$

$$\ddot{y} = c_4\dot{x} + c_5\dot{y} + c_6\dot{z}$$

$$\ddot{z} = c_7\dot{x} + c_8\dot{y} + c_9\dot{z}$$

where $c_1 = -75252.637$, $c_2 = 75923.605$, $c_3 = 23382.477$, $c_4 = 55410237.279$, $c_5 = -55904286.630$, $c_6 = -17217053.022$, $c_7 = -341023.007$, $c_8 = 344063.641$ and $c_9 = 105962.571$.

For the standardized data using the pre-computed derivatives, it was not possible to find equations that described the correct orbit.

5.4.2 Multiple Trajectories

With non-standardized data, both with the pre-calculated derivatives and without them, it is not possible to find equations for an orbit with any optimizer. Using standardized data it also was not possible to get any good results.

5.5 Noise Analysis: Robustness to heavy-tailed noise

Table 5.1: Robustness of SINDy to multiple types of noise - Custom Library.

Noise Type	Position	Velocity
Gaussian Noise	0.1%	1%
Laplacian Noise	0.1%	0.1%
Cauchy Noise	0.001%	0.01%

Table 5.2: Robustness of SINDy to multiple types of noise - Polynomial Library.

Noise Type	Position	Velocity
Gaussian Noise	0.37%	0.79%
Laplacian Noise	0.38%	0.53%
Cauchy Noise	0.01%	0.1%

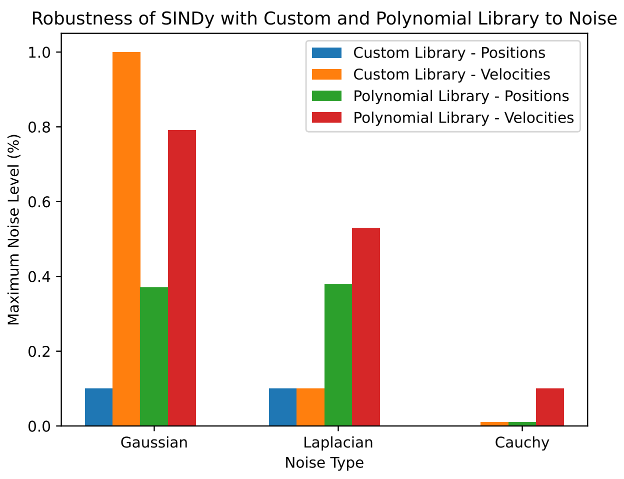


Figure 5.7: Robustness of SINDy to Different Types of Noise

To see how capable SINDy is in discovering PDEs in multiple noise conditions, considering that its performance excelled in discovering first-order PDEs, we introduced multiple types of noise to the data and then used SINDy to discover those first-order equations to check how robust it is, given a noisy input using the custom library function and the polynomial library whose results are shown respectively in Table 5.1 and 5.2. There were three types of noise we tested SINDy with due to their intrinsic characteristics. The first one we tested with was the Gaussian noise. One of the biggest advantages is that when dealing with a lot of data, it tends to respect the central limit theorem that the Gaussian distribution describes. Having thin tails and the majority of its probability mass around the mean also represents having a lower probability of generating outliers. Afterward, we applied two other different types of noise: Laplacian and Cauchy. The distributions that describe these two types of noise are heavy-tailed, so they are usually used for modeling outliers as they appear in the distribution areas further away from the mean. Laplacian noise can be considered two exponential distributions pointy around the mean whereas the Cauchy distribution has even heavier tails to the point where a mean value does not even parameterize it. One important step to remember is that the noise may have a different effect depending on which variables are affected, whether the noise is present in the positions or the velocities. We calculated the norm of a position vector and applied the different types of noise with a standard deviation varying from 0 to 10% of the norm of that vector and did the same for the velocities. The noise robustness associated with both the custom and polynomial libraries, encompassing the three distinct noise types and their impact on the positions and velocities, are depicted in Figure 5.7.

Using the **custom library**, we utilized SINDy to discover first-order equations in the presence of noise. For **Gaussian noise**, if there is only noise on the velocity, SINDy supports up to 1% noise before yielding non-explanatory equations for the system. On the other hand, if there is only noise on the position, SINDy supports up to 0.1% of noise. Regarding 1% noise on the velocity, SINDy does not support any noise on the position. However, with 0.1% noise on the velocity, SINDy supports as low as 10^{-6}% noise on the position.

Regarding **Laplacian noise**, SINDy demonstrates similar robustness in handling noise in the positions, but this ability decreases with velocities. If there is only noise on the velocities, SINDy allows up to 0.1% noise compared to the 1% tolerance with Gaussian noise. Similarly, if there is only noise on the positions, SINDy allows up to 0.1% noise, consistent with the Gaussian noise scenario.

Concerning **Cauchy noise**, it has two parameters, *loc* which specifies where the peak of the distribution will be on the X axis (which is 0 by default) and a scale parameter represents half the width of the PDF at half the maximum height. Fixing the position parameter and varying the scaling factor, having only noise on the positions, SINDy supports noise up to 0.001% of the position vector norm. For velocities, it supports noise up to 0.01% of the norm of a velocity vector, which is 1 order of magnitude less.

Considering the **polynomial library**, which has shown promising results, evaluating its performance in the presence of noise is crucial. The analysis in this section focuses on standardized data. For **Gaussian noise**, if there is only noise on the positions, SINDy supports a standard deviation up to 0.37% of the norm of the first position vector, corresponding to a noise level of up to 26 km in positions, which is reasonable. Regarding velocities, SINDy supports up to 0.79%, equivalent to 0.06 km/s. When the standard deviation for the positions is 26 km, the same percentage of noise is supported for velocities (0.79%).

Results for **Laplacian noise** are similar, with support for up to 27 km noise in positions and up to 0.04 km/s noise in velocities when noise is present only in the respective variables. In the case of maximum supported position noise (27 km), the supported noise in velocities remains at 0.04 km/s.

Lastly, for **Cauchy noise**, SINDy supports up to 0.01% of the norm of the positions and 0.1% of the norm of the velocities.

Comparing the three types of noise, it is evident that SINDy demonstrates greater robustness to noise on the velocities than on the positions. Additionally, it is important to note that when using the polynomial library with non-standardized data, SINDy struggles to identify any PDEs for the velocities, regardless of the introduced noise level. This emphasizes the importance of standardizing the data before applying SINDy to achieve more reliable results.

6

Predictive Methods for State Vectors

Our main objective is to improve the accuracy of predicting satellite positions for the future. We identified that the drag force, specifically the atmosphere density variable, contributes to the stochasticity in these predictions. To address this, we will first explore some methodologies, focusing on predicting future positions, as it is at its core a time-series forecasting problem. Subsequently, we will extend our investigation to predict the atmosphere density.

For these methods, we examined various techniques. We started by using the equations discovered with SINDy and propagated them to predict future positions, as explained earlier. Additionally, we briefly experimented with LSTM units, a type of recurrent neural network suited for time-series data [10, 19, 78]. We also tested feed-forward neural networks and a Physically-informed Neural Network (PINN) with multiple architectures in the data-driven part for performance comparison.

Through these baseline methodologies, we aim to evaluate their effectiveness in addressing the challenges posed by the drag force and atmosphere density variable. By comparing their results, we seek to identify their strengths and weaknesses in reducing predictive errors for satellite position forecasts.

In the subsequent sections, we will provide an analysis of the outcomes from each methodology. By understanding their performances in predicting future positions and velocities, we aim to identify promising approaches for further improvement and advance data-driven modeling for satellite trajectory prediction. This exploration of baseline methods will enrich our understanding and contribute to more accurate and reliable predictive models for satellite motion.

6.1 SINDy

Our analysis in the previous chapter showed us that when training the model with multiple orbits and employing the uncovered equations we managed to get errors that exhibited a desirable behavior, as they neither followed an exponential trend nor significantly deviated from the sinusoidal waves characterizing the motion along each axis. This encouraging outcome underscored the potential of the approach in predicting satellite positions for the future. However, when attempting to uncover equations with SINDy, we encountered one requirement: the data fed to the model had to be precise, with little noise. If the data had any noise beyond the tolerable levels analyzed in the noise analysis of the previous chapter (see 5.5), the subsequent propagation of equations would result in orbits that no longer conformed to the expected behavior. Instead, they might exhibit peculiar and undesirable characteristics, such as spiraling upwards or downwards into infinity.

This sensitivity to data precision in the SINDy approach highlighted a crucial limitation and demonstrated the importance of acquiring high-quality, noise-free measurements for achieving accurate predictions. The requirement for precise data becomes particularly relevant in scenarios where noise and uncertainties are prevalent, such as in real-world satellite tracking and trajectory estimation. Therefore, in practice, careful consideration and measures for data acquisition and preprocessing become imperative to ensure the success and reliability of the SINDy methodology.

6.2 LSTM Network

In the context of SINDy, as noise in the data inevitably increases, it becomes apparent that SINDy-derived equations may eventually deviate from describing any form of orbit. In contrast, when utilizing models with a data fidelity step, although the predictions might not be entirely accurate, they retain an essential characteristic: the predictions will still correspond to trajectories that resemble orbits, rather than deviating into non-orbital behaviors. This distinction shows the importance of incorporating data fidelity steps in maintaining the fundamental behavior of the predictions when dealing with the complexities introduced by noise in the input data.

Regarding these methods and subsequent architectures, the data input is structured in windows rather than the conventional dataframe format. This adaptation stems from the inherent nature of our task, which is time-series prediction. This is also in order to turn this forecasting into a supervised learning problem. Dealing with time-series data involving multiple variables, our objective is to predict these variables into the future. The windowed data approach enables prediction of the subsequent F timesteps based on the prior P timesteps. Each window incorporates crucial parameters: the input width (considering preceding timesteps), the label width (predicting future timesteps), and the shift (determining window overlap).

Regarding the LSTM network, it exhibited several limitations that warranted brief exploration. Among these is the extensive training time, averaging around 20 minutes per epoch even for moderate-sized datasets. In terms of performance, examining the error graph for a variable showed errors surpassing 500 km without additional noise, which is much higher than with SINDy.

However, the primary drawback of the LSTM network surfaced when used to predict the following timestep based on the prior 200 timesteps, this being very limiting. Errors in this scenario were considerable, revealing the limited capability of the model. Moreover, when the LSTM network was used for recursive prediction as done in [77], aiming to forecast multiple future timesteps, errors increased exponentially. This rendered the model impractical for generating any kind of accurate predictions in this problem.

In the upcoming sections, we will explore other network architectures and methodologies, focusing on their comparative performances. Through systematic evaluation, we aim to uncover approaches better suited to overcoming noise and uncertainties, ultimately advancing satellite trajectory prediction techniques.

6.3 Feed Forward Network

For the feed forward networks we experimented with three possible architectures.

First Architecture:

- Input Layer used to select the input steps from the input tensor and reshape it to [Batch Size, Input steps, Features].
- 3 Dense layers with 32, 16 and 32 neurons and LeakyReLU as an activation function.
- Flattening layer to make it [Batch Size, Output steps*Features]
- Reshaping layer to [Batch Size, Output Steps, Features] so that we restore the multi-dimensional structure and have the predicted features for the desired number of output steps.

Second Architecture:

- Input Layer used to select the input steps from the input tensor and reshape it to [Batch Size, Input steps, Features].
- 3 Dense layers with 32, 8 and 32 neurons and LeakyReLU as an activation function.
- Flattening layer to make it [Batch Size, Output steps*Features]
- Reshaping layer to [Batch Size, Output Steps, Features] so that we restore the multi-dimensional structure and have the predicted features for the desired number of output steps.

Third Architecture:

- Input Layer used to select the input steps from the input tensor and reshape it to [Batch Size, Input steps, Features].
- 3 Dense layers with 16, 8 and 16 neurons and LeakyReLU as an activation function.

- Flattening layer to make it [Batch Size, Output steps*Features]
- Reshaping layer to [Batch Size, Output Steps, Features] so that we restore the multi-dimensional structure and have the predicted features for the desired number of output steps.

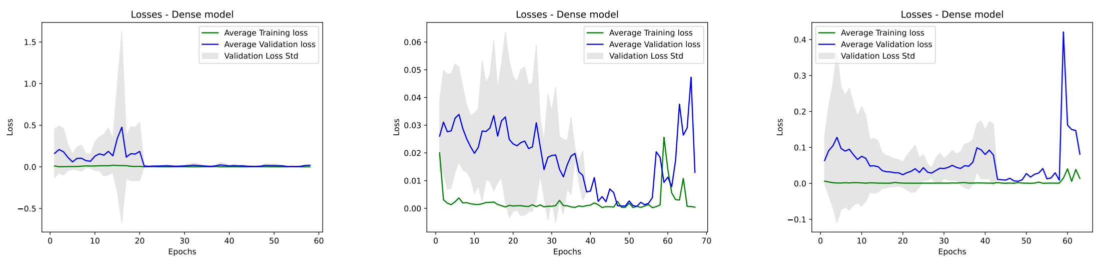
Overall, these architectures take as input a time series sequence of shape [Batch Size, Input Steps, Features] and apply three dense layers with LeakyReLU activation functions to extract higher-level representations. The output is reshaped to match the desired output shape of [Batch Size, Output Steps, Features], representing the predicted future time steps.

The tests we are going to make are based on how changing the features of the windows affect the predictions made and the respective errors between the multiple architectures.

6.3.1 Input Width = 1000, Output Width = 30

It is important to note that the time units that define the windows are in minutes. This means that in this case we are considering the values of the variables from the last 1000 minutes to predict the values for the following 30 minutes.

Firstly, we will look at the training and validation losses over the epochs for the multiple networks ([32, 16, 32], [32, 8, 32] and [16, 8, 16]) while running 10 trials for each configuration so that the values are more stable.



(32, 16, 32) Architecture (32, 8, 32) Architecture (16, 8, 16) Architecture

Figure 6.1: Loss Evolution for Different Architectures and 1000 Input Steps

Looking at the losses behaviours over the multiple networks, it is noticeable that on the least complex network [16, 8, 16], the average validation loss values decrease up to around epoch 50 and then start to increase due to overfitting as it is seen in Figure 6.1. When looking at the second network, the validation loss values seem to be lower than on the least complex network while also having smaller standard deviation values, proving it has more stability on the training and validation of this model. Finally, looking at the more complex network, the validation loss values seem to be the lowest ones, reaching practically zero at one point and not increasing from there onwards. However, there was one peak on the standard deviation over the 10 trials that turned out to be the highest recorded standard deviation value.

Secondly we will briefly look at the differences between the predicted values and the real values for each of the axis. As it is possible to see, the mean errors for the positions in X is 824.81 km for the first network, 341.20 km for the second network and 316.44 km for the third network. For the positions in Y it is around 272.76 km for the first network, 92.30 km for the second network and 136.30 km for the third network. For the positions in Z, these around 822.63 km for the first network, 365.09 km for the second network and 397.89 km for the third network. For the velocities in X the error was around 0.82 km/s for the first network, 0.36 km/s for the second network and 0.38 km/s for the third network. In Y was around 0.19 km/s for the first network, 0.08 km/s for the second network and 0.14 km/s for the third network. In Z they were around 0.84 km/s for the first network, 0.36 km/s for the second network and 0.30 km/s for the third network.

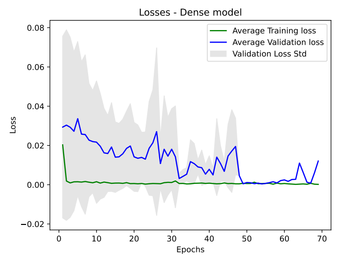
Analyzing the results obtained using an input window of 1000 minutes and outputting 30 minutes revealed that the two least complex networks, [16, 8, 16] and [32, 8, 32], yield significantly lower mean errors compared to the same variables on the most complex network.

The explanation on why the least complex networks provide better results in this specific problem could be attributed to two possible factors. Firstly, the simplicity of the architectures of the networks might be a key factor. In some cases, a more complex network with a larger number of parameters can lead to overfitting, especially when the dataset is relatively small which is the case. The simpler network, with its reduced complexity, is better suited to capture the essential patterns and generalize effectively. Secondly, the characteristics of the dataset itself could also play a role. Since it is relatively straightforward with sinusoidal waves and lacks intricate complex patterns, simpler models like these can avoid overfitting and perform well without the need for excessive complexity. These factors emphasize the importance of understanding the dataset and finding the right balance between model complexity and dataset characteristics to achieve optimal results.

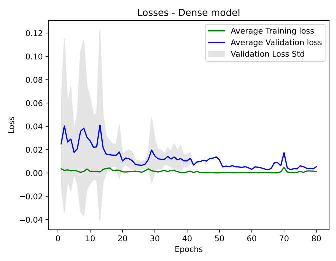
6.3.2 Input Width = 500, Output Width = 30

In this subsection, we explore the impact of using a smaller input window of 500 minutes instead of 1000 while predicting the same number of time-steps into the future. The idea behind this experiment is that reducing the input window size might lead to inferior results compared to the previous experiment. The expectation is that with a shorter input window, the model will have access to less historical information, potentially limiting its capacity to capture long-term dependencies and patterns in the data. As a result, the predictive performance of the model could be compromised. However, conducting this experiment allows us to assess the significance of the input window size on the performance of the model and gain insights into the trade-offs between input window length and prediction accuracy.

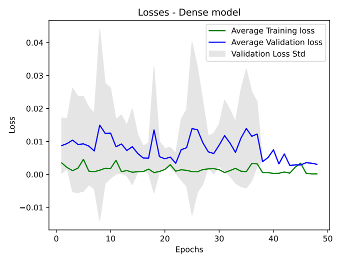
We will now look at the training and validation losses over the epochs for the multiple networks ([32, 16, 32], [32, 8, 32] and [16, 8, 16]).



(32, 16, 32) Architecture



(32, 8, 32) Architecture



(16, 8, 16) Architecture

Figure 6.2: Loss Evolution for Different Architectures and 500 Input Steps

Looking at the losses obtained, it is worth noting that the results obtained suggest a similar trend to that observed in the previous experiment but not quite the same. Concerning the most complex network, after the 60th epoch, it is possible to see the validation loss increasing representing overfitting which did not happen when the input window consisted of 1000 steps. This means that when having less data, this kind of network with more neurons per layer tends to overfit due to being too complex for this specific problem and the amount of historical data used. Looking at the losses behaviors of the other two simpler networks, it is important to note that in this case there was no overfitting, showing that simpler networks work better for when we have less historical data to deal with.

Looking now at the actual errors relative to the predicted values, the mean errors attained for the positions in X were 1693.12 km for the first network, 1350.41 km for the second network and 862.74 km for the third network. For the positions in Y, the errors were 277.96 km for the first network, 138.15 km for the second network and 60.70 km for the last network. For the positions in Z the errors were 1312.62 km for the first network, 1012.85 km for the second network and 628.41 km for the third network. Concerning the velocities, the errors for the X axis were 1.30 km/s for the first network, 1.03 km/s for the second network and 0.64 km/s for the third network. For the Y axis the errors were 0.41 km/s for the first network, 0.20 km/s for the second network and 0.088 km/s for the third network. Finally, for the Z axis the errors were 1.68 km/s for the first network, 1.31 km/s for the second network and 0.83 km/s for the last network.

Analyzing the results obtained using an input window of 500 minutes and outputting 30 minutes revealed once again that the least complex networks, [16, 8, 16] and [32, 8, 32], result in lower mean errors. Looking at these errors, it is possible to see that for all of the variables, the results are better on the least complex one. This could be due to the fact of having a smaller input window and the model is favoring a simpler architecture to make better predictions.

Upon comparing the results of the simpler network with the experiment involving the larger input window, it becomes evident that, in this case, the outcomes were inferior for almost all variables (apart from the position in Y on the most simple network and the velocity in Y on the second network). This aligns with our initial expectation that reducing the amount of available past information would lead to poorer predictions compared to when we have a more comprehensive understanding of the previous data. The premise holds that having less information about the past restricts the capability of the model to capture crucial patterns, consequently impacting its predictive accuracy.

6.4 PINN

After investigating the other methods, what was left was to use the technique that we initially considered which are PINNs.

Concerning its architecture, it differs from a fully connected network in the sense that here we have one large network consisting of two networks as it is possible to see in 6.3. The first one corresponds to a typical feed forward that can be tuned to have whichever width and depth values. The input of this network corresponds to what we are differentiating in respect to, which is time as well as other historical state variables we have, which in our case correspond to positions and velocities. The output of this network correspond to the predicted state vectors consisted of 6 variables for the number of timesteps we want to forecast. After this first network, its outputs will be inputs for the second network which correspond to the physical part. This network will act as a regularizer for the inputs that were fed by minimizing the residuals of the PDEs that govern the system.

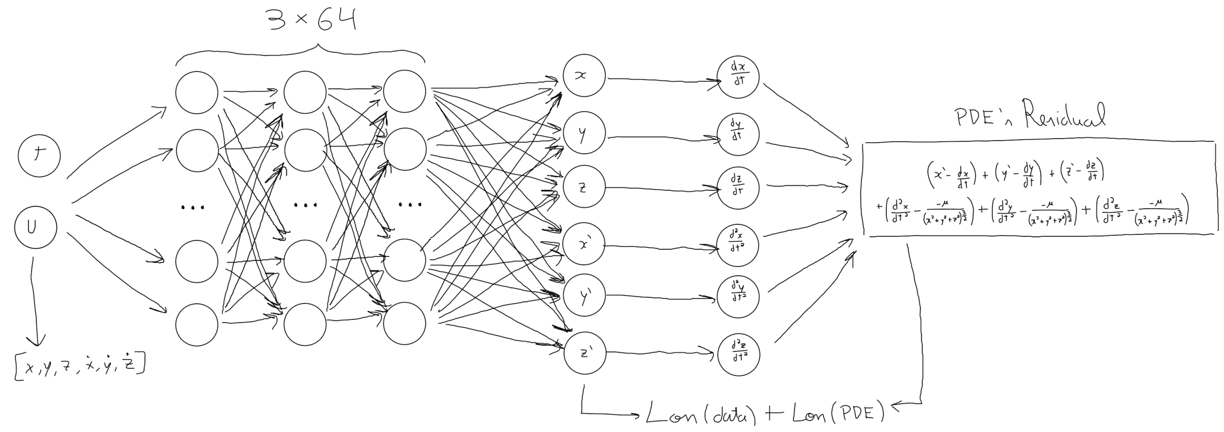

3 × 64
t
U
[x, y, z, ẋ, ẏ, ż]
x
y
z
x`
y`
z`
dx/dt
dy/dt
dz/dt
d²x/dt²
d²y/dt²
d²z/dt²
PDE'n Residual
(x` − dx/dt) + (y` − dy/dt) + (z` − dz/dt) + (d²x/dt² − −μ/(x²+y²+z²)^(3/2)) + (d²y/dt² − −μ/(x²+y²+z²)^(3/2)) + (d²z/dt² − −μ/(x²+y²+z²)^(3/2))
Lon(data) + Lon(PDE)

Figure 6.3: PINN Diagram

The loss function of this network differs from a typical loss function only consisting of a data-fidelity term. Here we add a second term concerning of how well the data respects the equations which in this case corresponds to how well the predicted state vectors respect the 6 partial differential equations that describe the changes in positions and velocities over the 3 axes (predicted outputs against the physical equation outputs). The equations used will be (1.1) which are the same as used previously.

For the PINN, we will be testing with the same architectures that we used without the physical part in order to assess which one performs better.

6.4.1 Unbounded Coefficients

One possible approach to the PINNs loss function is to not define any coefficients manually for each of the losses of the networks, which will result in both the data fidelity term and the residuals term being considered equally important without any explicit relative weighting. This approach allows the network to treat both terms on an equal footing during the optimization process. By not assigning specific weights, the network does not prioritize one term over the other based on pre-defined criteria.

However, it is important to consider the implications of not assigning manual weights. Without explicit relative weighting, the behavior of the network may be influenced by various factors, such as the scale of the loss values or the actual optimization process. Depending on these factors, the network may implicitly assign different priorities to the data fidelity and residuals terms, potentially leading to biased or suboptimal results.

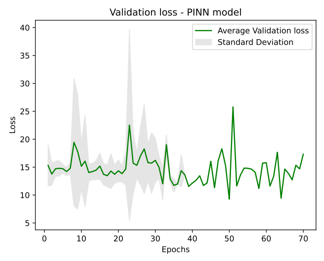
6.4.1.1 Input Width = 1000, Output Width = 30

When examining the losses of the networks trained with an input width of 1000 and an output width of 30, some interesting observations can be made. Firstly, it is apparent that the behaviors of all three networks are quite similar, with the losses hovering around the value of $\sqrt{15}m^2$ (due to the loss being the MSE added with the residuals). This similarity suggests that the networks are able to capture the underlying patterns and trends in the data to a comparable degree.

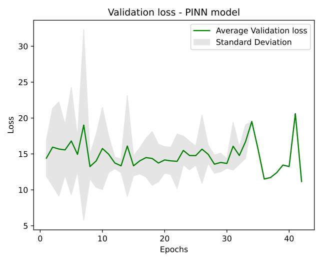
However, upon closer inspection, it becomes evident that the more complex network, with its architecture of [32, 16, 32], exhibits higher standard deviation values and appears to be less stable compared to the two simpler networks. The increased variability in the loss values implies that the complex network might struggle to consistently converge to an optimal solution, possibly due to the larger number of parameters and the added complexity introduced by the deeper layers.

Despite the higher variability, it is worth noting that the complex network still achieves a comparable overall performance in terms of the loss metric. This suggests that while the simpler networks may offer a more stable and predictable training process, the complex network is capable of capturing the underlying dynamics of the problem, albeit with some fluctuations in its optimization trajectory.

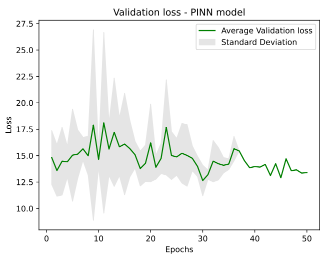
Furthermore, it is important to consider the trade-off between model complexity and generalization. While the simpler networks may exhibit more stable behavior during training, they might also have limitations in capturing intricate features and nuances in the data. The more complex network, although exhibiting higher variability, has the potential to learn more intricate patterns and generalize better to unseen data, provided that the variability does not translate into significant overfitting.



(32, 16, 32) Architecture

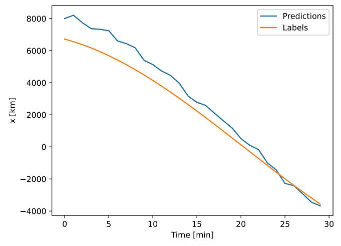


(32, 8, 32) Architecture

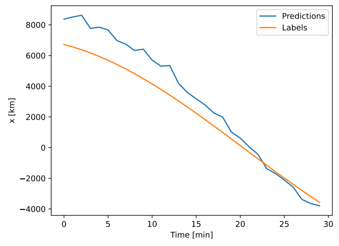


(16, 8, 16) Architecture

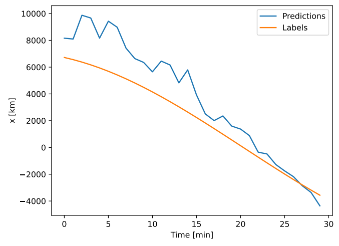
Figure 6.4: PINN - Loss Evolution for Different Architectures and 1000 Input Steps



(32, 16, 32) Architecture

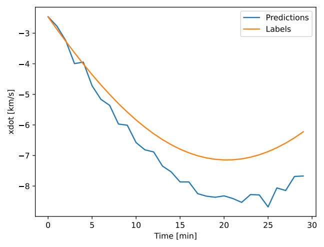


(32, 8, 32) Architecture

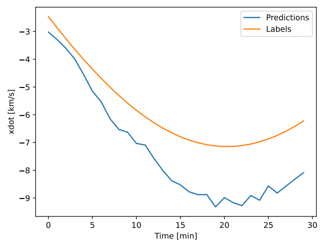


(16, 8, 16) Architecture

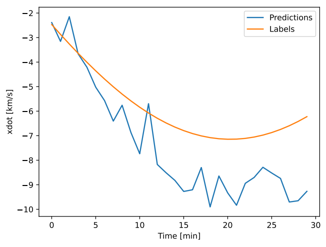
Figure 6.5: PINN - Predictions (x) for Different Architectures and 1000 Input Steps



(32, 16, 32) Architecture



(32, 8, 32) Architecture



(16, 8, 16) Architecture

Figure 6.6: PINN - Predictions $\dot{x}$ for Different Architectures and 1000 Input Steps

Let us now delve into the actual predictions made by the networks and assess their performance. To illustrate the results, we will focus on the predictions of positions and velocities in the X dimension. This choice allows us to examine the performance of the network in capturing two different variables that are inherently in different scales, providing valuable insights into its aptness to handle diverse aspects of the problem. By visualizing the predictions, we gain a clearer understanding of how well the networks capture the underlying dynamics of the system. For the positions in X, we can observe whether the networks accurately predict the trajectory of the object or if there are significant deviations from the ground truth. Likewise, for the velocities in X, we can assess how effectively the networks capture the speed of the object over time.

Taking a closer look at the predictions made by each of the networks, it becomes evident that the absence of any weight balancing between the terms in the loss function has a significant impact on the results. Without explicitly defining weights for the data fidelity term and the residuals term, the most complex network exhibits a distinct advantage, yielding the most favorable predictions compared to the other network architectures.

However, it is crucial to consider the underlying reasons behind this observed dominance. The absence of manual weight assignment allows the network to prioritize one term over the other based solely on the inherent characteristics of the problem. In cases where one term carries more significant importance or contributes more significantly to the overall loss, the network may inadvertently neglect or underemphasize the other term, leading to potential inaccuracies or inconsistencies in the predictions.

Taking a look at the predictions made with each of the networks, we can see that with no weights balancing both of the terms, the most complex network prevails and gives out the best results when compared to the other architectures. Due to not having any manually defined weights, the network can overshadow and neglect one of the terms in the loss function if the other one is much more important which is why it is a good idea to balance out both.

The performance evaluation of the three networks provides valuable insights into their predictive power across the six variables. Let us delve into the mean errors exhibited by each network for positions in X, Y, and Z, as well as the velocity components.

The most complex network showed mean errors of 778.52 km for positions in X, 34.46 km for positions in Y, 860.44 km for positions in Z and 0.87 km/s for velocity in X, 0.050 km/s for velocity in Y, 0.76 km/s for velocity in Z. The second network resulted in mean errors of 1076.45 km for positions in X, 172.70 km for positions in Y, 1373.03 km for positions in Z and 1.43 km/s for velocity in X, 0.16 km/s for velocity in Y, 1.10 km/s for velocity in Z. The most simple network resulted in a mean error of 1597.60 km for positions in X, 299.23 km for positions in Y, 1216.78 km for positions in Z and 1.59 km/s for velocity in X, 0.16 km/s for velocity in Y, 1.37 km/s for velocity in Z.

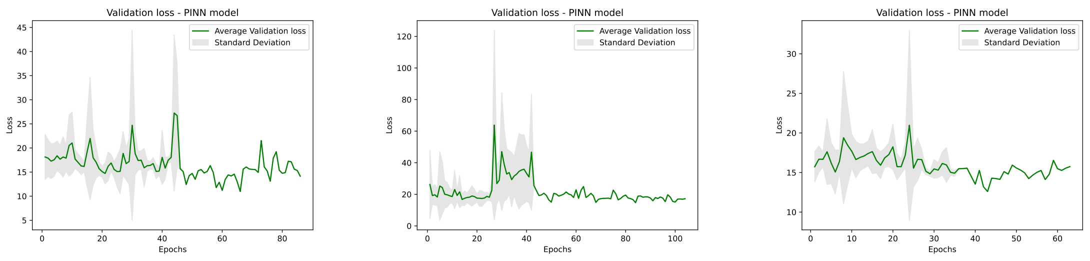
From these errors, we can conclude that in this case, the most complex network, despite its higher architectural complexity, consistently outperforms the other networks across all variables. It exhibits significantly lower mean errors for positions in X, Y, and Z, as well as for velocity components. This suggests that the intricate architecture of the network enables it to capture and model the underlying relationships more effectively, resulting in more accurate predictions.

6.4.1.2 Input Width = 500, Output Width = 30

Starting by looking at the losses, it is noticeable that the behaviors of all of the three networks are similar (hovering at around $\sqrt{20}m^2$ for the two more complex ones and slightly lower for the more simple one at around $\sqrt{15}m^2$. What is possible to see here is that the standard deviation values for all of the networks are much higher (specially on the first two) which represents more instability on the training of said models.

The higher standard deviation values for the more complex networks may be attributed to the fact that we want the model to forecast the variables for the same amount of timesteps as previously but only considering half of the input timesteps.

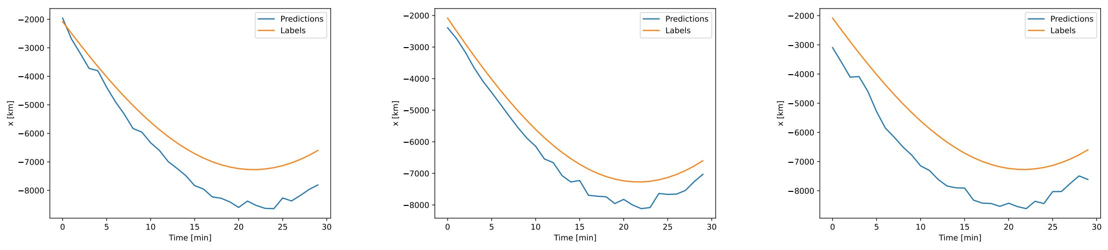
In contrast, the simpler network displays a lower standard deviation, implying a more stable training process. This could be to the fact that a simpler network works better when we are dealing with less input data. This proves the importance of striking a balance between model complexity and training stability, while having in mind our specific case.



(32, 16, 32) Architecture (32, 8, 32) Architecture (16, 8, 16) Architecture

Figure 6.7: PINN - Loss Evolution for different architectures and 500 Input Steps

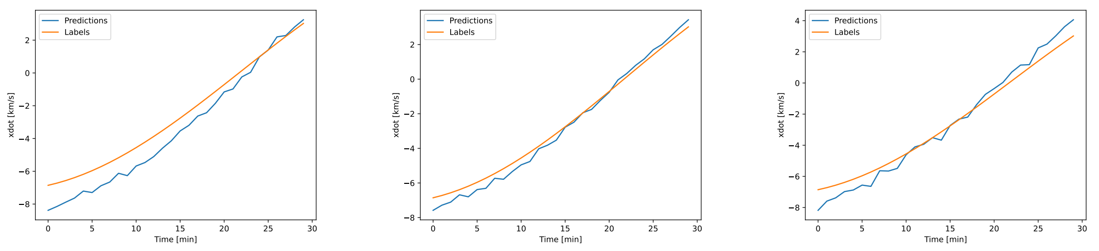
Let us delve into the predictions made by the networks and explore how the utilization of a smaller input window impacts the performance of the PINNs. To provide a comprehensive analysis, we will once again employ graphical visualization, focusing on the variables of interest, namely the positions (x) and velocities ($\dot{x}$). Considering the smaller input window size, it is anticipated that the networks may encounter challenges in comprehensively capturing the temporal dynamics of the system. The limited historical context provided by a shorter input window may hinder the comprehension of the network on understanding the evolving patterns and relationships between inputs and outputs over longer time intervals.



(32, 16, 32) Architecture (32, 8, 32) Architecture (16, 8, 16) Architecture

Figure 6.8: PINN - Predictions (x) for Different Architectures and 500 Input Steps

Examining the predictions made by each network, it becomes apparent that the second network outperforms the others in terms of predictive accuracy for both positions and velocities. The improved performance of the second network suggests that the reduced temporal context does not significantly hinder its ability to capture essential features of



(32, 16, 32) Architecture (32, 8, 32) Architecture (16, 8, 16) Architecture

Figure 6.9: PINN - Predictions ($\dot{x}$) for Different Architectures and 500 Input Steps

the system dynamics. This outcome emphasizes the capacity of the network to adapt and make effective use of the available information, even when operating under constraints imposed by a smaller input window. It is all a matter of finding the perfect network for the data we have.

Finally, lets analyze the mean errors attained with this setup. Concerning the most complex network, the mean errors for the positions in X were 895.34 km, 178.86 for positions in Y and 777.12 for positions in Z. For the velocities, the errors were 0.83 km/s for the velocity in X, 0.16 km/s for the velocity in Y and 0.86 km/s for the velocity in Z. For the second network, the mean errors for the positions in X were 563.10 km, for the positions in Y they were 83.85 km and for the positions in Z they were 285.80 km. Concerning the velocities, these were 0.32 km/s in the X axis, 0.13 km/s in the Y axis and 0.53 km/s in the Z axis. Finally, for the most simple network, the mean errors were 1231.84 km for the positions in X, 135.95 km for the positions in Y and 539.22 km for the Z axis. For the velocities, the errors were 0.50 km for the X axis, 0.23 km for the Y axis and 1.18 for the Z axis.

To conclude this section concerning of not using any coefficients for both terms, the choice of input window size has a notable impact on the predictive performance of the networks. When using 1000 input steps, the most complex network achieved the lowest mean errors for both positions and velocities. However, with a reduced input window of 500 steps, the second network demonstrated superior performance across all variables.

6.4.2 Bounded Coefficients

To establish a balanced relationship between the data fidelity term and the residuals term in the PINN loss function, it is essential to define coefficients that represent the relative importance of each term. By assigning appropriate weights, we can precisely control the contribution of each term and achieve the desired balance. In this section, we will explore the use of different weight combinations for both the 1000 and 500 input step scenarios, allowing for a comparative analysis of the results obtained.

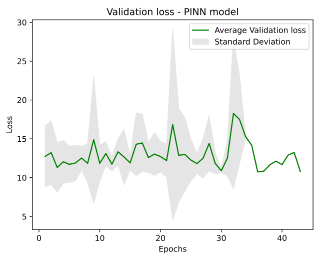
By varying the weights, we can investigate the impact of different emphasis placed on the data fidelity and residuals terms. Determining the perfect balance between the data fidelity and residuals terms requires careful consideration and experimentation. It is important to note that the optimal setup may depend on various factors, such as the actual problem domain, the dataset, as well as network architecture thus fine-tuning the weights based on empirical evaluation is crucial to achieve the best performance.

The loss corresponds to a linear combination between the data fidelity and the PDE residuals such as $weight * Data\_Fidelity + (1 - weight) * PDE\_residuals$. What we will be experimenting is to explore different $weight$ coefficients over the multiple architectures.

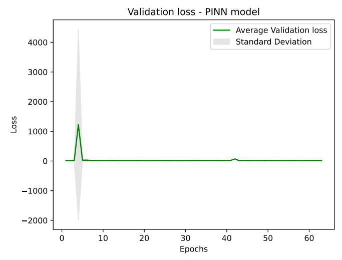
6.4.2.1 Input Width = 1000, Output Width = 30

To begin the analysis, we consider a weight combination where 10% of importance is assigned to the data fidelity term, while the remaining 90% of importance is allocated to the residuals of the equations. This weight distribution aims to emphasize the significance of capturing the underlying system dynamics while still considering the available data. This section is dedicated to using an input window with 1000 steps.

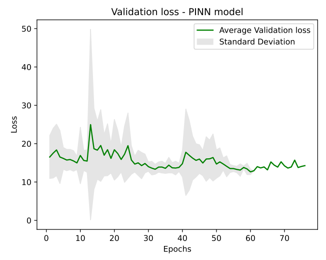
Upon examining the losses obtained using this weight combination, we observe that the values continue to hover around $\sqrt{15}$ to $\sqrt{20}\ m^2$ for all three networks. However, an interesting pattern emerges when analyzing the second network. Across multiple trials, the models exhibit substantial variation in their losses, particularly around the fifth epoch. This suggests that the training process for the second network may be more sensitive to the weight distribution, potentially leading to fluctuations in its performance.



(32, 16, 32) Architecture



(32, 8, 32) Architecture



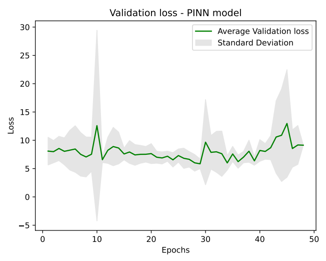
(16, 8, 16) Architecture

Figure 6.10: PINN - Loss Evolution for Different Architectures, 1000 Input Steps and 0.1 Weight

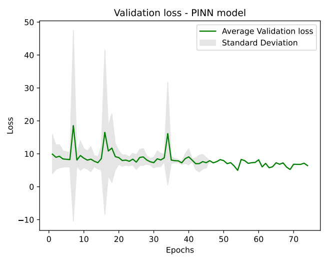
Comparing the losses obtained from the perfectly balanced weight distribution with the previous experiment, where the weight was highly biased towards the physical part, reveals some interesting observations. Firstly, the loss values achieved using the balanced weight distribution are noticeably lower, hovering around $\sqrt{10}m^2$. This indicates that assigning equal importance to both the data fidelity term and the residuals of the equations yields improved performance in terms of minimizing the overall loss.

Furthermore, examining the standard deviation values of the losses provides additional insights into the stability and consistency of the networks. It is evident that the standard deviation values are reduced across all three networks when using the balanced weight distribution. This reduction implies that the models exhibit more consistent and reliable performance throughout the training process.

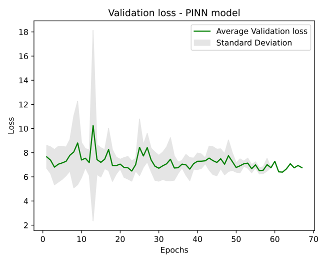
While the improvements in loss values and standard deviations are apparent for all networks, it is worth noting that the second and third networks demonstrate more well-defined and smaller standard deviation values. This suggests that the balanced weight distribution has a particularly beneficial effect on these architectures, resulting in more stable training and enhanced predictive capabilities.



(32, 16, 32) Architecture



(32, 8, 32) Architecture



(16, 8, 16) Architecture

Figure 6.11: PINN - Loss Evolution for Different Architectures, 1000 Input Steps and 0.5 Weight

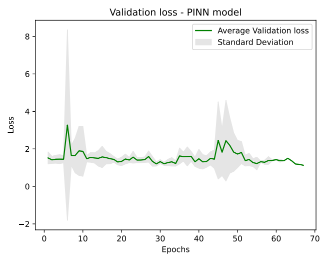
In the final weight distribution scenario, where the data part is assigned a weight of 0.9 and the residuals a weight of 0.1, some observations can be made. Firstly, both the loss values and the standard deviations are significantly smaller compared to the previous experiments. This suggests that emphasizing the data fidelity term while downplaying the residuals leads to improved overall performance in terms of minimizing the loss.

The smaller loss values obtained in this scenario indicate that the predictions are aligning more closely with the given data. By assigning a higher weight to the data part, the network is effectively prioritizing the fitting of the observed data points, resulting in a better alignment between the predicted values and the actual measurements. Consequently, the overall loss is reduced, indicating a higher level of accuracy and precision in the predictions of the network.

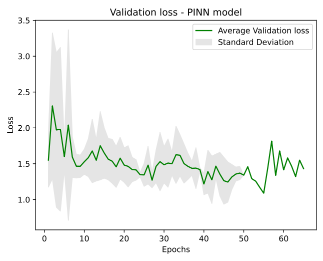
Similarly, the smaller standard deviations reflect a higher level of consistency and stability in the training process. The reduced variability in the loss values across multiple trials signifies that the performance of the network is more reliable and less prone to fluctuation. However, it is worth noting that while the standard deviations are smaller overall, the most complex network exhibits higher deviations compared to the other two architectures.

Repeating the process conducted in the experiment with unbounded coefficients, we will now examine the actual predictions generated by the models using the different weight distributions.

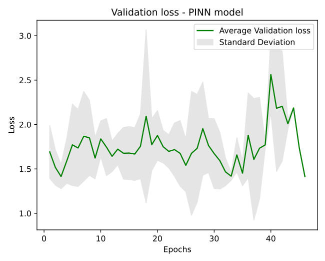
Firstly, analyzing the results attained, concerning the positions in x, when we consider 0.1 importance for the data, we can see that the results are similar between the multiple networks. For the velocity in X, the same is seen. Shifting our attention to weighting



(32, 16, 32) Architecture

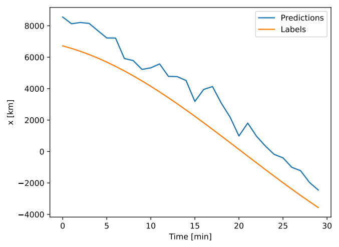


(32, 8, 32) Architecture

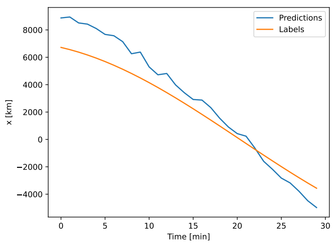


(16, 8, 16) Architecture

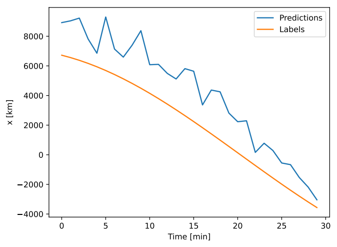
Figure 6.12: PINN - Loss Evolution for Different Architectures, 1000 Input Steps and 0.9 Weight



(32, 16, 32) Architecture

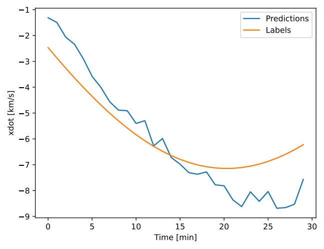


(32, 8, 32) Architecture

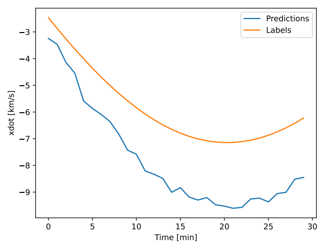


(16, 8, 16) Architecture

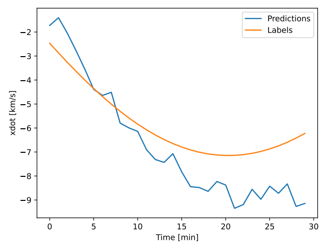
Figure 6.13: PINN - Predictions (x) for Different Architectures, 1000 Input Steps and 0.1 Weight



(32, 16, 32) Architecture



(32, 8, 32) Architecture

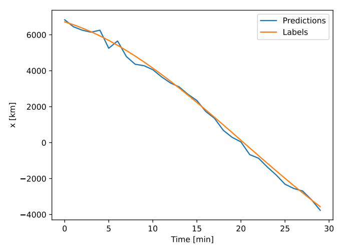


(16, 8, 16) Architecture

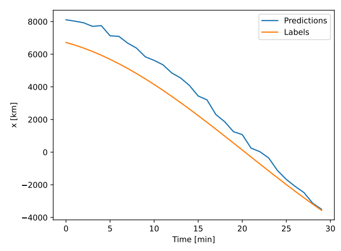
Figure 6.14: PINN - Predictions ($\dot{x}$) for Different Architectures, 1000 Input Steps and 0.1 Weight

equally both terms, we can see that the best results, both for the position and velocity, are with the most complex network [32, 16, 32]. When considering 0.9 importance to the data part, we can also see that the results are better with the most complex network.

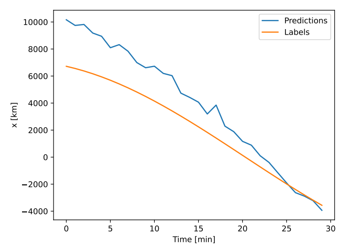
When examining the actual mean errors for each of the networks, it is possible to visualize the differences that having a different setup have in the predictions. Let us begin with the most complex network, where a weight of 0.1 was assigned. In this scenario, the network achieved mean errors of 1554.78 km for positions in x, 102.48 km for positions in y, and 789.59 km for positions in z. In terms of velocities, the errors were 0.91 km/s in the x-axis, 0.14 km/s in the y-axis, and 1.54 km/s in the z-axis.



(32, 16, 32) Architecture

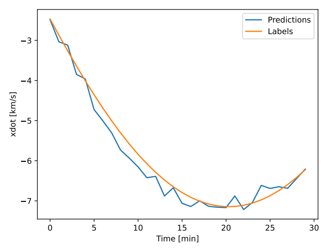


(32, 8, 32) Architecture

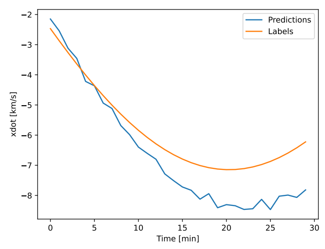


(16, 8, 16) Architecture

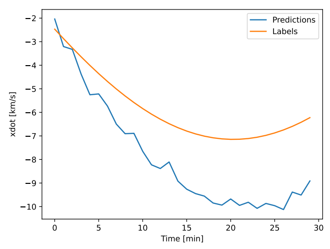
Figure 6.15: PINN - Predictions (x) for Different Architectures, 1000 Input Steps and 0.5 Weight



(32, 16, 32) Architecture

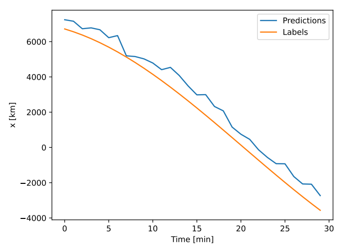


(32, 8, 32) Architecture

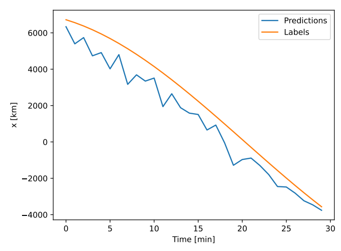


(16, 8, 16) Architecture

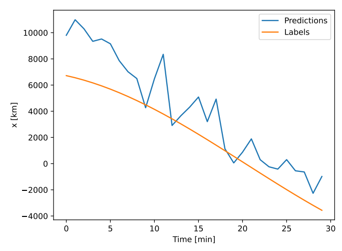
Figure 6.16: PINN - Predictions ($\dot{x}$) for Different Architectures, 1000 Input Steps and 0.5 Weight



(32, 16, 32) Architecture



(32, 8, 32) Architecture

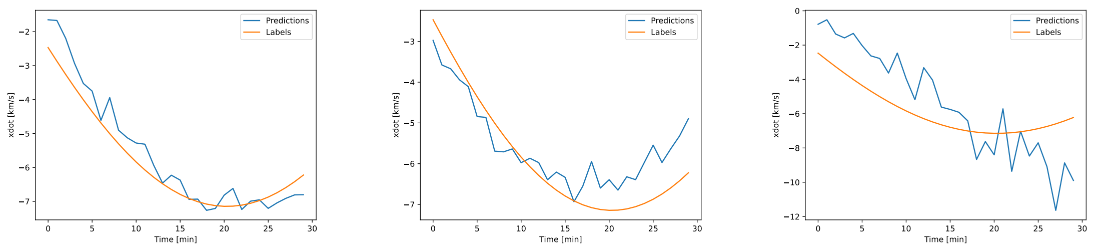


(16, 8, 16) Architecture

Figure 6.17: PINN - Predictions (x) for Different Architectures, 1000 Input Steps and 0.9 Weight

However, when the weight was adjusted to 0.5, a remarkable improvement was observed. The mean errors significantly decreased, with values around 181.78 km for positions in x, 133.34 km for positions in y, and 171.78 km for positions in z. The velocity errors were also reduced to 0.18 km/s in the x-axis, 0.076 km/s in the y-axis, and 0.20 km/s in the z-axis.

For the weight distribution of 0.9, the most complex network encountered a different outcome. While the mean errors decreased compared to the initial weight of 0.1, they were still higher than when using the weight of 0.5. The errors amounted to 722.23 km for positions in x, 201.38 km for positions in y, and 458.97 km for positions in z. In terms



(32, 16, 32) Architecture (32, 8, 32) Architecture (16, 8, 16) Architecture

Figure 6.18: PINN - Predictions ($\dot{x}$) for Different Architectures, 1000 Input Steps and 0.9 Weight

of velocities, the errors were 0.43 km/s in the x-axis, 0.11 km/s in the y-axis, and 0.86 km/s in the z-axis.

Looking at the second network, the pattern of improvement with the weight adjustment is repeated. With a weight of 0.1, the network exhibited mean errors of 1220.26 km for positions in x, 187.04 km for positions in y, and 1940.04 km for positions in z. The velocity errors were 1.90 km/s in the x-axis, 0.27 km/s in the y-axis, and 1.13 km/s in the z-axis. However, when the weight was adjusted to 0.5, significant improvements were observed. The mean errors decreased to 1078.03 km for positions in x, 73.52 km for positions in y, and 766.77 km for positions in z. The velocity errors reduced to 0.82 km/s in the x-axis, 0.10 km/s in the y-axis, and 1.00 km/s in the z-axis. Similarly, for the weight distribution of 0.9, the second network displayed improved performance but with slightly higher errors compared to the weight of 0.5. The mean errors were 916.58 km for positions in x, 140.95 km for positions in y, and 636.68 km for positions in z. The velocity errors were 0.56 km/s in the x-axis, 0.29 km/s in the y-axis, and 0.74 km/s in the z-axis.

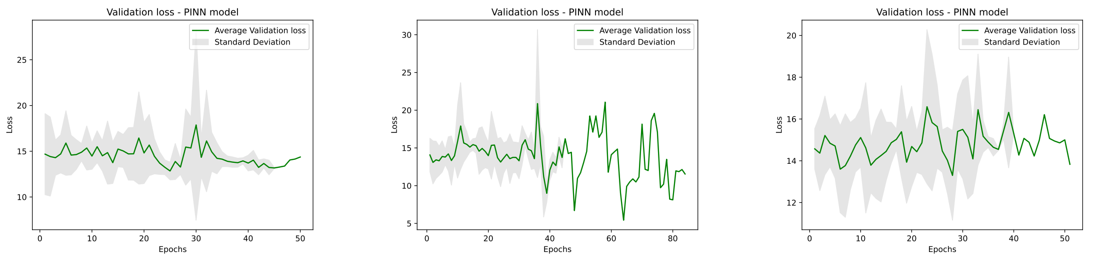
Lastly, we turn our attention to the most simple network. With a weight of 0.1, this network exhibited mean errors of 2121.29 km for positions in x, 191.59 km for positions in y, and 1249.36 km for positions in z. The velocity errors were 1.21 km/s in the x-axis, 0.30 km/s in the y-axis, and 2.38 km/s in the z-axis. With a weight adjustment to 0.5, the mean errors decreased to 1758.71 km for positions in x, 612.95 km for positions in y, and 1906.75 km for positions in z. The velocity errors were 2.040 km/s in the x-axis, 0.31 km/s in the y-axis, and 1.70 km/s in the z-axis. For the weight distribution of 0.9, the most simple network continued to display higher errors compared to the weight of 0.5. The mean errors were 2060.44 km for positions in x, 253.98 km for positions in y, and 2063.61 km for positions in z. The velocity errors amounted to 1.89 km/s in the x-axis, 0.36 km/s in the y-axis, and 2.13 km/s in the z-axis.

In conclusion, adjusting the weight distributions in the bounded coefficients experiment yielded varying effects on the mean errors of the networks. In general, the experiments showed that assigning a weight of 0.5 to the data part and 0.5 to the residuals yielded the best results across the networks. This balanced weight distribution consistently led to lower mean errors for both position and velocity predictions compared to other weight combinations. It strikes a good compromise between incorporating the available data and enforcing the underlying physical equations. The best setup showed up to be the most complex network using a weight of 0.5.

6.4.2.2 Input Width = 500, Output Width = 30

To delve into the analysis of the section that considers a 500-step input window, we start by examining a weight combination where 10% of importance is assigned to the data fidelity term, while the remaining 90% of importance is allocated to the residuals of the equations. This weight distribution aims to strike a balance between capturing the system dynamics and leveraging the available data. By focusing on a narrower input window, we can explore how the amount of historical data taken into account affects the performance of the PINNs.

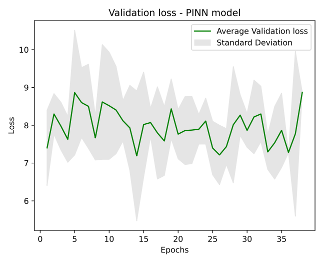
When examining the losses in this section, we observe that they consistently hover around $\sqrt{15}m^2$ for all three networks. This indicates a relatively stable performance across epochs. However, it is worth noting that during certain epochs, particularly for the second network, the validation losses drop below the $\sqrt{10}m^2$ threshold. This suggests that the ability of the network to capture the underlying dynamics and fit the data is particularly strong at those specific points in training. Additionally, the standard deviation values, which reflect the variability of the results across multiple trials, are fairly comparable among the three networks. This indicates a similar level of consistency in their performances.



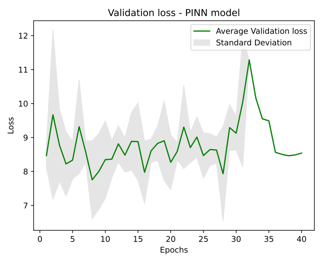
(32, 16, 32) Architecture (32, 8, 32) Architecture (16, 8, 16) Architecture

Figure 6.19: PINN - Loss Evolution for Different Architectures, 500 Input Steps and 0.1 Weight

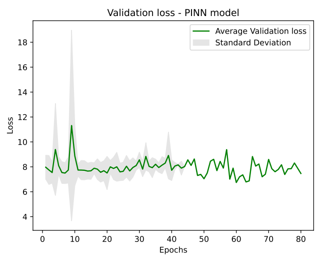
Moving on to the scenario where both the data fidelity term and the residuals are equally weighted, we observe notable improvements in the loss values across all network configurations. The losses decrease significantly and stabilize around $\sqrt{8}m^2$, indicating a more accurate and precise prediction performance. This balanced weight distribution allows the networks to effectively capture the dynamics of the system while still leveraging the available data. Additionally, the standard deviation values, which provide insights into the variability of results, also exhibit a decrease compared to the previous experiment where the physical part had higher importance. This reduction in standard deviation signifies a higher level of consistency and reliability in the predictions made by the networks, further underscoring the benefits of weight balance in achieving improved performance.



(32, 16, 32) Architecture



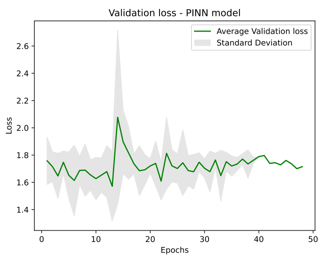
(32, 8, 32) Architecture



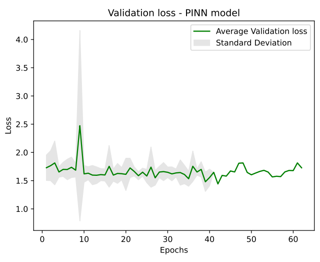
(16, 8, 16) Architecture

Figure 6.20: PINN - Loss Evolution for Different Architectures, 500 Input Steps and 0.5 Weight

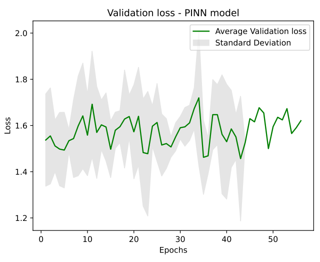
Similarly to the observations made with the larger input window, we find that increasing the weight assigned to the data fidelity term leads to further reductions in the loss values. Allocating 90% importance to this term results in significant improvements, with the loss values decreasing to approximately $\sqrt{2}m^2$. This weight distribution places a greater emphasis on accurately fitting the available data, allowing the networks to better capture the underlying patterns. The remarkable reduction in the loss values demonstrates the effectiveness of incorporating a higher weight for the data fidelity term in enhancing the overall predictive capabilities of the models.



(32, 16, 32) Architecture



(32, 8, 32) Architecture

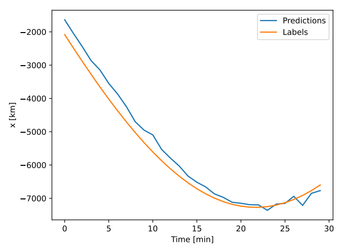


(16, 8, 16) Architecture

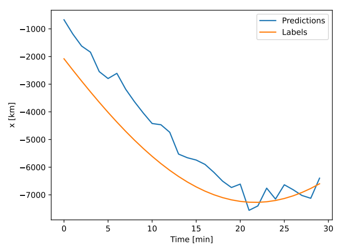
Figure 6.21: PINN - Loss Evolution for Different Architectures, 500 Input Steps and 0.9 Weight

Continuing our investigation, we will now revisit the experiment conducted with an input window of 1000 steps and shift our focus towards evaluating the concrete predictions produced by the models under varying weight distributions.

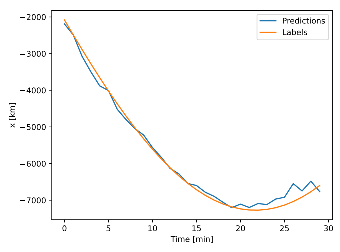
In examining the generated predictions, it is noteworthy to observe that the network with the simplest architecture demonstrates superior performance for the considered variables when assigning 90% importance to the residuals of the PDEs. This improvement observed with the reduced input dataset can be attributed to the inherent advantage of a simpler model in handling limited data. This finding underscores the importance of



(32, 16, 32) Architecture

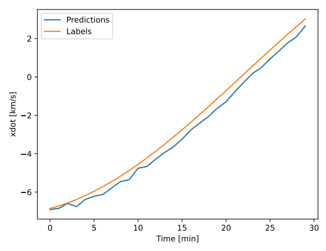


(32, 8, 32) Architecture

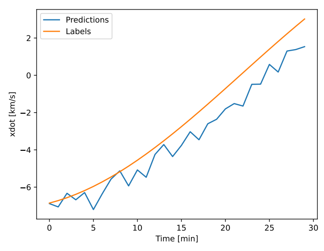


(16, 8, 16) Architecture

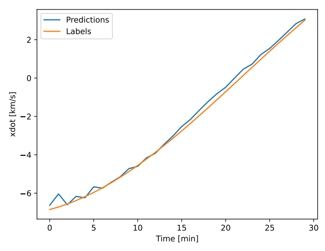
Figure 6.22: PINN - Predictions (x) for Different Architectures, 500 Input Steps and 0.1 Weight



(32, 16, 32) Architecture

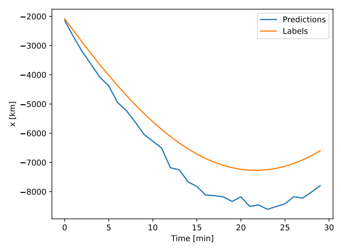


(32, 8, 32) Architecture

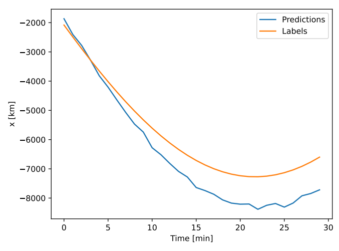


(16, 8, 16) Architecture

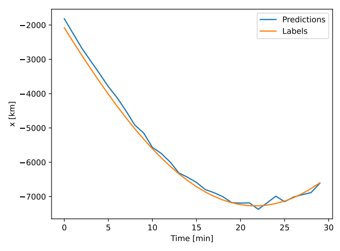
Figure 6.23: PINN - Predictions ($\dot{x}$) for Different Architectures, 500 Input Steps and 0.1 Weight



(32, 16, 32) Architecture



(32, 8, 32) Architecture

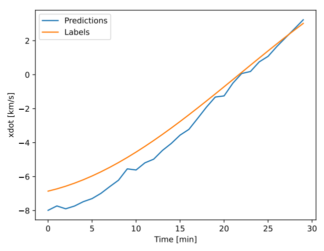


(16, 8, 16) Architecture

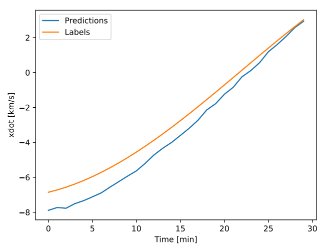
Figure 6.24: PINN - Predictions (x) for Different Architectures, 500 Input Steps and 0.5 Weight

identifying an optimal configuration specific to the problem at hand. Similarly, when both terms are assigned equal importance, the basic network consistently outperforms the other architectures, albeit with diminished disparities. However, it is important to note that when allocating 90% importance to the data term, the performance of all three networks is not desirable. This outcome can be attributed to the utilization of a smaller dataset, where the combination of reduced data availability and increased emphasis on it contributes to a degradation in predictive accuracy.

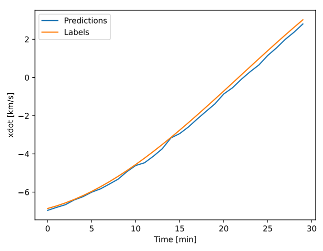
In conclusion, for this relatively simple dataset consisting mainly of sinusoidal waves, the most effective architectures are either a simpler dense network (16-8-16) with greater



(32, 16, 32) Architecture

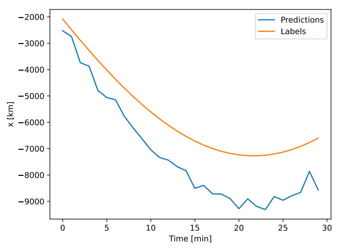


(32, 8, 32) Architecture

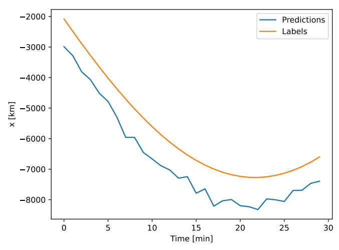


(16, 8, 16) Architecture

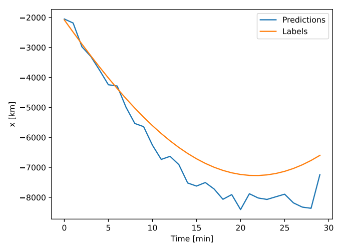
Figure 6.25: PINN - Predictions ($\dot{x}$) for Different Architectures, 500 Input Steps and 0.5 Weight



(32, 16, 32) Architecture

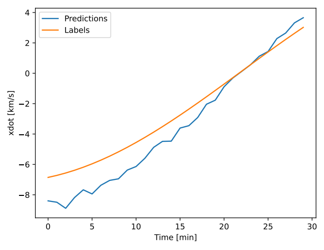


(32, 8, 32) Architecture

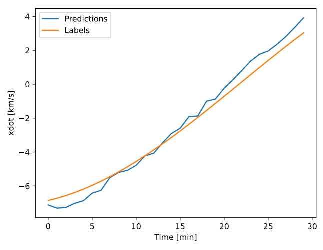


(16, 8, 16) Architecture

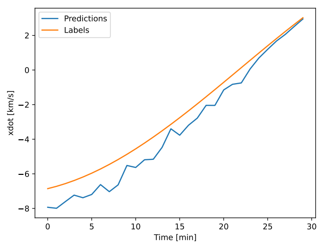
Figure 6.26: PINN - Predictions (x) for Different Architectures, 500 Input Steps and 0.9 Weight



(32, 16, 32) Architecture



(32, 8, 32) Architecture



(16, 8, 16) Architecture

Figure 6.27: PINN - Predictions ($\dot{x}$) for Different Architectures, 500 Input Steps and 0.9 Weight

emphasis on the physical component, or a more complex network with a larger input size and increased reliance on the available data (weight of 0.5 instead of 0.1). These architectures demonstrate improved performance in capturing the repetitive patterns present in the data over time. The findings highlight the importance of selecting an appropriate network architecture and weight distribution considering a dataset as input to achieve accurate predictions in this specific context.

These experiments have yielded significant insights into the advantages of incorporating domain-specific knowledge and exploiting the underlying physical equations within the context of predictive modeling. By comparing the performance of a dense neural network with the optimal configuration against that of a PINN, notable improvements have been observed.

The findings reveal that the PINN consistently outperforms the dense network in terms of positional predictions. The mean errors achieved by the PINN are consistently three times lower across the positions, highlighting its superior accuracy and ability to capture the underlying dynamics of the system. However, it is worth noting that the Y-axis predictions were slightly worse. This suggests the presence of additional complexities or variations in the Y-axis dynamics that may require further investigation.

Concerning the velocities, the PINN exhibits enhanced performance across all axes. The mean errors obtained by the PINN are significantly lower than those of the dense network, indicating its capability to capture the relationships between the position and velocity variables. This improvement in velocity predictions can be attributed to the ability of PINNs to incorporate the governing equations and exploit the known physical constraints, leading to more accurate and consistent results.

These findings underscore the importance of leveraging domain-specific knowledge and incorporating physics-based constraints in a predictive scenario like this one. The ability of PINNs to integrate physical laws into the learning process enables it to capture the dynamics of the system more effectively.

7 Comparing SINDy with PINNs

In this chapter, we aim to conduct a comparative analysis of the two methods previously analyzed, namely SINDy and PINNs. By directly comparing these methods, we seek to draw conclusions regarding their respective advantages and performance characteristics. Both SINDy and PINNs offer promising paths for data-driven modeling and and variable prediction, but they differ in their underlying principles and methodologies.

We will investigate various aspects of these methods, including their ability to accurately identify system dynamics, their computational efficiency, and their robustness in the presence of noise. By comparing their performances on the same set of problems with the same datasets, we aim to gain a deeper understanding of the strengths and weaknesses inherent in each approach. To ensure a fair and robust comparison, we will carefully design our experimental setup, considering factors such as dataset selection, training procedures, hyperparameter tuning, and performance metrics. Through experimentation, analysis, and interpretation of results, we aim to uncover the unique advantages, limitations, and performance characteristics of these methods. The insights derived from this study will not only contribute to this work but to also guide researchers in different domains in making decisions regarding the choice of method for data-driven modeling of complex dynamical systems.

When comparing the robustness to noise between SINDy and PINNs, one important detail outstands. One of the main disadvantages of SINDy is its limited ability to handle high levels of noise in the observations of the system. While SINDy has shown great performance under moderate noise levels, surpassing the levels tested in previous studies can significantly hinder its effectiveness. When applied to a dataset with excessive noise, SINDy may fail to identify the underlying equations governing the system. This limitation renders SINDy less reliable and potentially useless in scenarios where the measurements are highly noisy since the predictions for the future are based on propagating the found equations.

In contrast, PINNs offer a more robust alternative in the presence of significant noise. The initial phase of PINNs involves leveraging the actual data points available to the network, employing a mean squared error loss to compare the predictions of the network with the corresponding labels. This initial step does not strictly rely on the explicit incorporation of governing equations into the network architecture. The second part that corresponds to the residuals however, acts as a regularizer for the predictions made and does not rely strictly on them as it happens with SINDy. As a result, PINNs can better deal with noisy measurements and provide reasonable predictions, even when the system dynamics are not accurately captured by explicit equations. This characteristic makes PINNs an appealing choice in scenarios where the measurements exhibit a high degree of noise.

By highlighting these differences in robustness to noise, it becomes evident that the choice between SINDy and PINNs depends on the specific characteristics of the data, particularly the noise levels present in the observations. While SINDy excels in situations with relatively low noise, it may struggle to provide any predictions when faced with substantial noise since no equations are found in this scenario. In contrast, PINNs can offer more reliable results in the presence of significant noise, as they can leverage the data directly without strictly relying on explicit equations.

Analyzing the actual results obtained from using SINDy, it becomes evident that it performs better both in in-distribution and out-of-distribution scenarios when predicting the future positions and velocities of a satellite compared to the PINNs.

Firstly, the performance of SINDy on an out-of-distribution orbit deserves attention. The experiments ran in Chapter 5 where SINDy was trained with more than one orbit were tested on an out-of-distribution orbit. Despite the inherent challenges posed by predicting future states based on data that deviates from the training distribution, SINDy exhibits notable robustness. The results obtained from out-of-distribution orbits demonstrate that SINDy maintains a high level of accuracy, suggesting its capability to generalize well beyond the training data. This generalization ability is valuable in real-world scenarios where system dynamics can vary, ensuring the applicability and reliability of the SINDy approach. The errors attained for an out-of-distribution with SINDy were around the same values than with the predicitons for an in-distribution orbit with the PINNs which shows that SINDy is the obvious choice in this case.

In order to improve the predictive performance of the PINN, an obvious choice was to train the model using data from multiple orbits. By incorporating data from different orbits, the model would have access to a more comprehensive representation of the space surrounding the Earth. This expanded dataset has the potential to enhance the ability of the model to capture complex relationships and make more accurate predictions.

To accommodate the multi-orbit data, a 3D representation was adopted. In this representation, the data was organized as a cube, where the first dimension corresponded to the sequence length, representing the number of points in the timeseries. The second dimension represented the number of different orbits, allowing the model to consider data from various satellites performing different trajectories. Finally, the third dimension corresponded to the features used in the model, such as positions, velocities and atmosphere density.

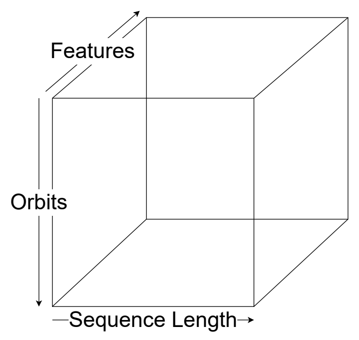


Figure 7.1: Data Format

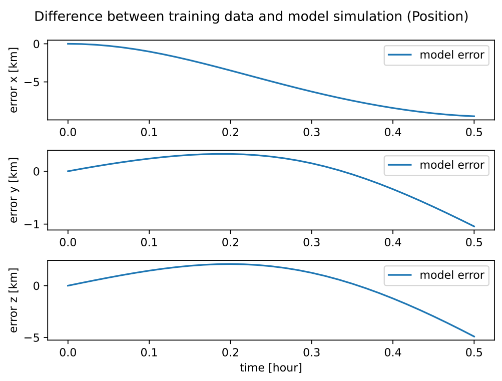
Figure 7.1 visualizes this data format as a cube. Each timestep across all satellites and their respective variables could be evaluated within this cube structure. By incorporating data from multiple orbits, the model could potentially leverage the diversity of orbital characteristics and capture the intricate dynamics of the space environment more effectively.

However, a significant challenge arises when dealing with this expanded data format, namely, the growth in the number of parameters. As the size of the input window increases, the number of parameters required by the network also increases. Consequently, the computational feasibility of the model becomes a crucial consideration. With hundreds of millions of parameters, training the model using input windows longer than a certain threshold, such as 5 minutes, becomes computationally intractable.

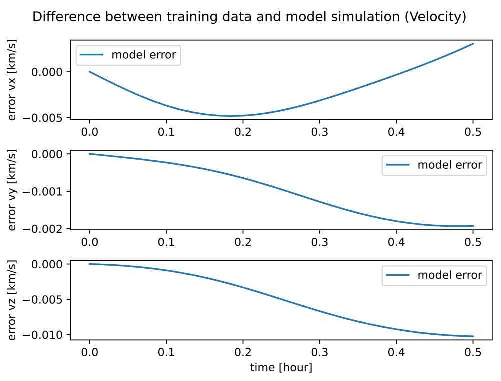
When examining the performance of SINDy on an in-distribution orbit and predicting thirty minutes into the future, the results demonstrate exceptional accuracy as depicted in Figure 7.2. The errors between the predicted values and the corresponding actual labels are minimal, indicating that SINDy effectively captures the underlying dynamics of the system. These promising results highlight the capability of SINDy to accurately model and predict the behavior of the satellite for this time window. Comparing these results with PINNs, using one with the best setup found trained with only one orbit, testing its predictions for the future for the same orbit (in-distribution) the errors averaged at around 100 km for the positions which are much higher than with SINDy.

The predictions made with SINDy for future positions and velocities showcase outstanding accuracy across all axes. The mean errors are as follows: for position predictions, 4.79 km in the X-axis, 0.0028 km in the Y-axis, and 0.22 km in the Z-axis. Regarding velocity predictions, the mean errors are 2.09×10^{-3} km/s in the X-axis, 9.88×10^{-4} km/s in the Y-axis, and 5.01×10^{-3} km/s in the Z-axis. These impressive results affirm the precision and reliability of SINDy in capturing and forecasting the ephemerides of the satellite.

The impressive results obtained from SINDy for both in-distribution and out-of-distribution orbits further emphasize its potential for predicting the positions and velocities of satellites. These findings demonstrate the strengths and advantages of the SINDy approach in accurately capturing system dynamics and making predictions, even



Error in Positions



Error in Velocities

Figure 7.2: SINDy - Errors for Positions and Velocities over time

in challenging scenarios up to a certain level of noise.

When we analyzed the results attained with the PINNs, we reached the conclusion that they were subpar, even when comparing with such a simple model as SINDy. However, current research is being done on the poor performance of PINNs due to the optimization process and the difficulties that arise when posing soft constraints on the predictions made [34, 72, 68, 70, 11, 30].

8 Atmosphere Density

Up until now, our focus has been on predicting the positions and velocities of satellites. However, in order to gain a comprehensive understanding of the behavior of the satellite and its interaction with the surrounding environment, it is crucial to consider the prediction of another vital variable, atmosphere density.

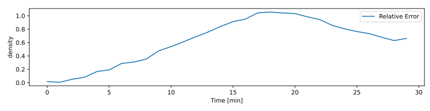
The scarcity of measurements of atmosphere density, along with its significant impact on the positions and velocities of satellites, reveals the importance of accurately predicting this variable [7]. The density of the atmosphere plays a pivotal role in determining the magnitude of forces acting on the satellite, such as drag and atmospheric disturbances. These forces, in turn, directly influence the trajectory of the satellite, speed, and overall orbital behavior.

Considering the limited availability of direct measurements for atmosphere density, leveraging data-driven approaches that incorporate domain knowledge in any way, like SINDy and PINNs do, becomes valuable. These methods have demonstrated their effectiveness in capturing complex relationships from sparse or noisy data, making them well-suited for inferring and predicting atmosphere density based on the available measurements. By extending our analysis to encompass the prediction of atmosphere density, we intend to understand the interplay between this variable and the movement of the satellite.

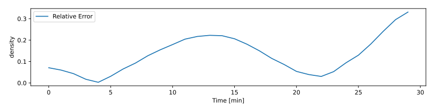
8.1 PINN

Given that the data used in this study was generated using a propagator that already considered external factors affecting the positions and velocities of the satellite, we realized that the original architecture of the PINNs did not incorporate these effects, specifically atmospheric drag, in the equations used for residual minimization. This posed a significant discrepancy between the generated data and the equations being optimized in the PINNs framework. To address this discrepancy, it became necessary to incorporate the term related to atmospheric drag within the PINNs architecture. By doing so, the expression for minimizing the residual between the predicted values and the observed data could better align with the physical reality that is implictly in the dataset.

After successfully incorporating the missing variables into the dataset, the next step is to repeat the experiments to now predict the atmosphere density. Two different architectures were tested for predicting the atmosphere density which were the best ones from the previous section. The first architecture was a simpler network with a structure of 16-8-16 trained with 500 input steps and assigned weights of 0.1 for the data-driven part and 0.9 for the residuals.The second architecture was a more complex network with a structure of 32-16-32, trained with 1000 input steps and assigned a weight of 0.5 between the two terms. By implementing these architectures, the aim was to leverage the available dataset, including the newly incorporated variables, to accurately predict the atmosphere density. Let us look at the relative errors over time for both of these networks in Figure 8.1.



Network [32, 16, 32]



Network [16, 8, 16]

Figure 8.1: Relative Errors for both network architectures over a timespan of 30 minutes

Upon analyzing the relative errors over time for both chosen architectures, it is evident that they do not exhibit an ideal performance, although they are not entirely inaccurate. Particularly, the relative errors of the simpler network demonstrate some capability to capture the variations in density as they relate to the positions of the satellite throughout the duration. Despite the imperfections, the fact that the simpler network exhibits some accuracy between density variations and satellite positions is promising. It suggests that even with a relatively basic architecture, there is potential to capture essential patterns and dependencies within the data.

8.2 SINDy

In addition to exploring the neural network approach, another interesting approach for predicting the atmosphere density is to re-apply SINDy. The objective here is to uncover a PDE that accurately describes the behavior of the density variable, enabling us to propagate and obtain future values.

Looking at the results attained by this approach in Figure 8.2, this yielded remarkably superior results compared to those attained using the PINNs. By leveraging the potential of SINDy to identify governing equations from data, we were able to discover a PDE that effectively captures the dynamics of the atmosphere density.

One reason why SINDy achieves better results for the atmosphere density variable is its ability to handle multiple trajectories, which provides a distinct advantage over the

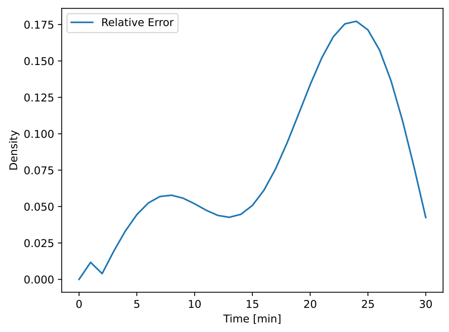


Figure 8.2: SINDy - Relative Errors for Density Predictions

PINN methodology. Unlike PINNs, which are limited to considering a single orbit due to computational constraints, SINDy seamlessly incorporates multiple trajectories without encountering intractability issues. The incorporation of multiple trajectories in SINDy offers one big benefit. Firstly, it allows for a more comprehensive understanding of the system by incorporating data from various regions around the Earth. These diverse measurements capture the dynamics of the atmosphere in different geographical locations, which inherently improves the predictions for the density variable since it is dependent on the positions. Despite the computational challenges faced by PINNs, the ability of SINDy to incorporate multiple trajectories highlights its flexibility and scalability. By leveraging data from diverse orbits, SINDy aggregates the information available from various orbits, leading to more accurate predictions of the atmosphere density variable.

9 Future Work

This research has provided valuable insights into the effectiveness of incorporating PINNs for predicting positional and velocity variables in a system characterized by sinusoidal wave-like behaviors. Building upon the current findings, there are several avenues for future work that can further enhance the understanding and applicability of PINNs in this context.

One potential direction for future research is to extend the current model to incorporate the prediction of additional variables, specifically a realistic approach of the atmosphere density. While the dataset used was enhanced with the atmosphere density value values used by the propagator which in turn uses the NRMLSIS model, incorporating realistic atmosphere density values in the PINN framework can provide more accurate and comprehensive predictions.

Since the data-centric side of the network already considers the intrinsic variations in the atmosphere density, if there were enough realistic measurements for the atmosphere density, an average value could be considered within the physical part of the network and what could be done is to minimize the difference between the predicted values by the first part of the network and the average density values considered. This approach can serve as a proxy for atmosphere density prediction and enable faster and more accurate predictions of both future state vectors and density.

Moreover, investigating the use of PINNs for predicting density alongside position and velocity variables offers the potential to overcome the limitations and accumulated errors often encountered in conventional atmosphere models. By leveraging the physical constraints embedded within the PINN architecture, the predictions can be made more robust and reliable, circumventing the exponential behaviors often observed in traditional models. This could lead to significant improvements in atmospheric modeling, particularly in systems where accurate density estimation is critical, such as within Neuraspace. Evaluating the performance of the network across different input sizes and architectures, and comparing it to other machine learning and physics-based modeling approaches, can further validate and refine the proposed methodology.

9.1 Conclusion

In conclusion, the future work should focus on extending the current PINN framework to incorporate a more realistic prediction of the atmosphere density alongside positional and velocity variables. It is also interesting to explore more techniques in the realm of PINNs to bypass the optimization difficulties we encountered which are also being investigated by other researchers [34, 72, 68, 70, 11, 30]. This approach has the potential to overcome the limitations of existing atmosphere models and enable faster, more accurate predictions.

However, it is still important to understand that we managed to describe the movement dynamics of satellites for both in and out-of-distribution orbits with such a simple model as SINDy proving the ability of using empirically found PDEs to describe their motion. The results obtained were accurate, highlighting the potential of such a simple and lightweight approach for improving our understanding of space systems. Nevertheless, there is still significant room for accuracy and predictive power improvement since the errors remain in the kilometers range. Further exploration of different function libraries and optimization techniques may lead to even better results.

Overall, these results offer valuable insights into the behavior of space systems and provide a promising foundation for future research in this area. With the continued development of SINDy and other machine learning techniques, we will likely see further improvements in our ability to model and predict the behavior of satellites or debris, ultimately contributing to better tracking of those objects.

Bibliography

[1] N. Aeronautics and S. Administration. *In-Space Propulsion*. url: https://www.nasa.gov/sites/default/files/atoms/files/4._soa_in-space_propulsion_chapter_2022_0.pdf (cit. on p. 2).

[2] N. Aeronautics and S. Administration. *Orbital Debris Quarterly News*. Jan. 2015. url: https://orbitaldebris.jsc.nasa.gov/quarterly-news/pdfs/odqnv19i1.pdf (cit. on p. 1).

[3] N. Aeronautics and S. Administration. *The Threat of Orbital Debris and Protecting NASA Space Assets from Satellite Collisions*. Apr. 2009. url: http://images.spaceref.com/news/2009/ODMediaBriefing28Apr09-1.pdf (cit. on p. 1).

[4] Y. Akahoshi et al. "Influence of space debris impact on solar array under power generation". In: *International Journal of Impact Engineering* 35.12 (2008). Hypervelocity Impact Proceedings of the 2007 Symposium, pp. 1678–1682. issn: 0734-743X. doi: https://doi.org/10.1016/j.ijimpeng.2008.07.048. url: https://www.sciencedirect.com/science/article/pii/S0734743X0800170X (cit. on p. 2).

[5] M. Z. Alom et al. "A State-of-the-Art Survey on Deep Learning Theory and Architectures". In: *Electronics* 8 (Mar. 2019), p. 292. doi: 10.3390/electronics8030292 (cit. on p. 12).

[6] B. Bowman et al. "A New Empirical Thermospheric Density Model JB2008 Using New Solar and Geomagnetic Indices". In: *AIAA/AAS Astrodynamics Specialist Conference and Exhibit*. doi: 10.2514/6.2008-6438. eprint: https://arc.aiaa.org/doi/pdf/10.2514/6.2008-6438. url: https://arc.aiaa.org/doi/abs/10.2514/6.2008-6438 (cit. on p. 8).

[7] S. Bruinsma et al. "Thermosphere and satellite drag". In: *Advances in Space Research* (2023). issn: 0273-1177. doi: https://doi.org/10.1016/j.asr.2023.05.011. url: https://www.sciencedirect.com/science/article/pii/S0273117723003587 (cit. on p. 66).

[8] B. F. Chao. "Earth's oblateness and its temporal variations". In: *Comptes Rendus Geoscience* 338.14 (2006). La Terre observée depuis l'espace, pp. 1123–1129. ISSN: 1631-0713. DOI: https://doi.org/10.1016/j.crte.2006.09.014. URL: https://www.sciencedirect.com/science/article/pii/S1631071306002690 (cit. on p. 7).

[9] J. G. Charles Bussy-Virat Aaron Ridley. "Effects of Uncertainties in the Atmospheric Density on the Probability of Collision Between Space Objects". In: *Space Weather* 16 (2018), pp. 519–537. DOI: https://doi.org/10.1029/2017SW001705 (cit. on p. 16).

[10] K. Chen, Y. Zhou, and F. Dai. "A LSTM-based method for stock returns prediction: A case study of China stock market". In: *2015 IEEE International Conference on Big Data (Big Data)*. 2015, pp. 2823–2824. DOI: 10.1109/BigData.2015.7364089 (cit. on p. 40).

[11] S. Cuomo et al. *Scientific Machine Learning through Physics-Informed Neural Networks: Where we are and What's next*. 2022. arXiv: 2201.05624 [cs.LG] (cit. on pp. 65, 70).

[12] H. D. Curtis. *Orbital Mechanics for Engineering Students*. Linacre House, Jordan Hill, Oxford OX2 8DP: Elsevier Butterworth-Heinemann, 2005. ISBN: 0-7506-6169-0 (cit. on pp. 2, 6–8, 29).

[13] B. C. Daniels and I. Nemenman. "Automated adaptive inference of phenomenological dynamical models". In: *Nature communications* 6.1 (2015), p. 8133 (cit. on p. 18).

[14] V. A. Davis et al. "Spacecraft Charging Interactive Handbook". In: *6th Spacecraft Charging Technology*. Nov. 1998, pp. 211–215 (cit. on p. 2).

[15] A. B. Downey. *Think Stats: Exploratory Data Analysis in Python*. O'Reilly Media, 2014, pp. 57–60. ISBN: 1491907339 (cit. on p. 24).

[16] K. J. Erickson. *What Is the Solar Cycle?* URL: https://spaceplace.nasa.gov/solar-cycles/en/ (cit. on p. 3).

[17] ESA. *Space debris by the numbers*. 2023. URL: https://www.esa.int/Space_Safety/Space_Debris/Space_debris_by_the_number (cit. on p. 21).

[18] G. Faulconbridge. *U.S. and Russia track satellite crash debris*. 2009. URL: https://www.reuters.com/article/us-space-collision-idUSTRE51A8IA20090212 (cit. on p. 2).

[19] R. Fu, Z. Zhang, and L. Li. "Using LSTM and GRU neural network methods for traffic flow prediction". In: *2016 31st Youth Academic Annual Conference of Chinese Association of Automation (YAC)*. 2016, pp. 324–328. DOI: 10.1109/YAC.2016.7804912 (cit. on p. 40).

[20] K. Fukami et al. “Sparse identification of nonlinear dynamics with low-dimensionalized flow representations”. In: *Journal of Fluid Mechanics* 926 (Sept. 2021). DOI: 10.1017/jfm.2021.697. URL: https://doi.org/10.1017%2Fjfm.2021.697 (cit. on p. 20).

[21] O. Fuks and H. A. Tchelepi. “LIMITATIONS OF PHYSICS INFORMED MACHINE LEARNING FOR NONLINEAR TWO-PHASE TRANSPORT IN POROUS MEDIA”. In: *Journal of Machine Learning for Modeling and Computing* 1.1 (2020), pp. 19–37. ISSN: 2689-3967 (cit. on p. 15).

[22] Z. Hao et al. *Physics-Informed Machine Learning: A Survey on Problems, Methods and Applications*. 2023. arXiv: 2211.08064 [cs.LG] (cit. on p. 4).

[23] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition*. Springer Series in Statistics. Springer New York, 2009. ISBN: 9780387848587. URL: https://books.google.pt/books?id=tVIjmNS3Ob8C (cit. on p. 19).

[24] D. Hathaway. “The Solar Cycle”. In: *Living Rev. Sol. Phys.* 39 (Sept. 2015), p. 227. DOI: 10.1007/lrsp-2015-4 (cit. on p. 3).

[25] D. Ivanov et al. “Decentralized differential drag based control of nanosatellites swarm spatial distribution using magnetorquers”. In: *Advances in Space Research* 67.11 (2021). Satellite Constellations and Formation Flying, pp. 3489–3503. ISSN: 0273-1177. DOI: https://doi.org/10.1016/j.asr.2020.05.024. URL: https://www.sciencedirect.com/science/article/pii/S0273117720303471 (cit. on p. 20).

[26] N. Johnson. “The Collision of Iridium 33 and Cosmos 2251: The Shape of Things to Come”. In: National Aeronautics and Space Administration. 60th International Astronautical Congress, 2010. URL: https://ntrs.nasa.gov/api/citations/20100002023/downloads/20100002023.pdf (cit. on p. 2).

[27] K. Kaheman, J. N. Kutz, and S. L. Brunton. “SINDy-PI: a robust algorithm for parallel implicit sparse identification of nonlinear dynamics”. In: *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences* 476.2242 (Oct. 2020). DOI: 10.1098/rspa.2020.0279. URL: https://doi.org/10.1098%2Frspa.2020.0279 (cit. on p. 20).

[28] A. Kaptanoglu. *How to effectively use the SINDy method for system identification*. 2021. URL: https://youtube.com/playlist?list=PLN90bHJU-JLoOfEk0KyBs2qLTV7OkMZ25 (cit. on p. 30).

[29] A. A. Kaptanoglu et al. “PySINDy: A comprehensive Python package for robust sparse system identification”. In: *Journal of Open Source Software* 7.69 (2022), p. 3994. DOI: 10.21105/joss.03994. URL: https://doi.org/10.21105/joss.03994 (cit. on pp. 18, 30).

[30] G. E. Karniadakis et al. “Physics-informed machine learning”. In: *Nature Reviews Physics* 3.6 (June 2021), pp. 422–440. ISSN: 2522-5820. DOI: 10.1038/s42254-021-00314-5. URL: https://doi.org/10.1038/s42254-021-00314-5 (cit. on pp. 4, 65, 70).

[31] K. Karniadakis. “Physics-informed machine learning”. In: *Nat Rev Phys* 3 (2021), pp. 422–440 (cit. on pp. 5, 10, 11).

[32] S. Khodairy et al. “Impact of solar activity on Low Earth Orbiting satellites”. In: *Journal of Physics: Conference Series* 1523 (Apr. 2020), p. 012010. DOI: 10.1088/1742-6596/1523/1/012010 (cit. on p. 3).

[33] D. P. Kingma and J. Ba. “Adam: A Method for Stochastic Optimization”. In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings* (2015). URL: http://arxiv.org/abs/1412.6980 (cit. on p. 15).

[34] A. Krishnapriyan et al. “Characterizing possible failure modes in physics-informed neural networks”. In: *Advances in Neural Information Processing Systems*. Ed. by M. Ranzato et al. Vol. 34. Curran Associates, Inc., 2021, pp. 26548–26560. URL: https://proceedings.neurips.cc/paper/2021/file/df438e5206f31600e6ae4af72f2725f1-Paper.pdf (cit. on pp. 13–15, 65, 70).

[35] K. Kusano et al. “A physics-based method that can predict imminent large solar flares”. In: *Science* 369.6503 (2020), pp. 587–591. DOI: 10.1126/science.aaz2511. eprint: https://www.science.org/doi/pdf/10.1126/science.aaz2511. URL: https://www.science.org/doi/abs/10.1126/science.aaz2511 (cit. on p. 3).

[36] P. Labs. *PLANET LABS PUBLIC ORBITAL EPHEMERIDES*. URL: https://ephemerides.planet-labs.com/ (cit. on p. 22).

[37] Z. K. Lawal et al. “Physics-Informed Neural Network (PINN) Evolution and Beyond: A Systematic Literature Review and Bibliometric Analysis”. In: *Big Data and Cognitive Computing* 6.4 (2022). ISSN: 2504-2289. DOI: 10.3390/bdcc6040140. URL: https://www.mdpi.com/2504-2289/6/4/140 (cit. on p. 4).

[38] H. W. Lilliefors. “On the Kolmogorov-Smirnov Test for the Exponential Distribution with Mean Unknown”. In: *Journal of the American Statistical Association* 64.325 (1969), pp. 387–389. DOI: 10.1080/01621459.1969.10500983. eprint: https://www.tandfonline.com/doi/pdf/10.1080/01621459.1969.10500983. URL: https://www.tandfonline.com/doi/abs/10.1080/01621459.1969.10500983 (cit. on p. 25).

[39] S. Maddu et al. “Inverse-Dirichlet Weighting Enables Reliable Training of Physics Informed Neural Networks”. In: *Machine Learning: Science and Technology* 3.1 (July 2021). DOI: 10.13140/RG.2.2.30690.04806 (cit. on pp. 14, 15).

[40] N. M. Mangan et al. *Inferring biological networks by sparse identification of nonlinear dynamics*. 2016. arXiv: 1605.08368 [math.DS] (cit. on p. 20).

[41] P. M. Mehta et al. "Satellite drag coefficient modeling for thermosphere science and mission operations". In: *Advances in Space Research* (2022). ISSN: 0273-1177. DOI: https://doi.org/10.1016/j.asr.2022.05.064. URL: https://www.sciencedirect.com/science/article/pii/S0273117722004458 (cit. on pp. 2, 3).

[42] J. Meseguer, I. Pérez-Grande, and A. Sanz-Andrés. "3 - Keplerian orbits". In: *Spacecraft Thermal Control*. Ed. by J. Meseguer, I. Pérez-Grande, and A. Sanz-Andrés. Woodhead Publishing, 2012, pp. 39–57. ISBN: 978-1-84569-996-3. DOI: https://doi.org/10.1533/9780857096081.39. URL: https://www.sciencedirect.com/science/article/pii/B9781845699963500034 (cit. on p. 2).

[43] T. S. Minna Palmroth Maxime Grandin. "Lower-thermosphere–ionosphere (LTI) quantities: current status of measuring techniques and models". In: *Annales Geophysicae* 39 (2021), pp. 189–237. DOI: https://doi.org/10.5194/angeo-39-189-2021 (cit. on p. 16).

[44] NASA. *Frequently Asked Questions: Orbital Debris*. 2011. URL: https://www.nasa.gov/news/debris_faq.html (cit. on p. 2).

[45] NASA. *Shield Development*. URL: https://hvit.jsc.nasa.gov/shield-development/ (cit. on p. 2).

[46] V. Nateghi, M. Manzi, and M. Vasile. "Autoencoder-based Thermospheric Density Estimation Using GPS Tracking Data". In: Oct. 2021 (cit. on p. 17).

[47] V. Nateghi, M. Manzi, and M. Vasile. "Autoencoder-based Thermospheric Density Estimation Using GPS Tracking Data". In: Oct. 2021 (cit. on p. 20).

[48] Navigation and A. I. Facility. *An Overview of Reference Frames and Coordinate Systems in the SPICE Context*. Apr. 2009. URL: https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/Tutorials/pdf/individual_docs/17_frames_and_coordinate_systems.pdf (cit. on p. 6).

[49] J. Pateras, P. Rana, and P. Ghosh. "A Taxonomic Survey of Physics-Informed Machine Learning". In: *Applied Sciences* 13.12 (2023). ISSN: 2076-3417. DOI: 10.3390/app13126892. URL: https://www.mdpi.com/2076-3417/13/12/6892 (cit. on pp. 4, 10).

[50] H. Peng and X. Bai. "Exploring Capability of Support Vector Machine for Improving Satellite Orbit Prediction Accuracy". In: *Journal of Aerospace Information Systems* 15.6 (2018), pp. 366–381. DOI: 10.2514/1.I010616. eprint: https://doi.org/10.2514/1.I010616. URL: https://doi.org/10.2514/1.I010616 (cit. on pp. 20, 21).

[51] J. M. Picone et al. “NRLMSISE-00 empirical model of the atmosphere: Statistical comparisons and scientific issues”. In: *Journal of Geophysical Research: Space Physics* 107.A12 (2002), SIA 15-1-SIA 15–16. DOI: https://doi.org/10.1029/2002JA009430. eprint: https://agupubs.onlinelibrary.wiley.com/doi/pdf/10.1029/2002JA009430. URL: https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/2002JA009430 (cit. on p. 8).

[52] J. L. Proctor et al. “Exploiting sparsity and equation-free architectures in complex systems”. In: *European Physical Journal Special Topics* 223 (Dec. 2014). DOI: 10.1140/epjst/e2014-02285-8 (cit. on p. 18).

[53] T. Pulkkinen. “Space Weather: Terrestrial Perspective”. In: *Living Reviews in Solar Physics* 4.1 (May 2007), p. 1. ISSN: 1614-4961. DOI: 10.12942/lrsp-2007-1. URL: https://doi.org/10.12942/lrsp-2007-1 (cit. on p. 3).

[54] PySINDy. *SSR and FROLs (the greedy algorithms!) examples*. 2022. URL: https://github.com/dynamicslab/pysindy/blob/master/examples/11_SSR_FROLS_examples.ipynb (cit. on p. 30).

[55] M. Raissi, P. Perdikaris, and G. E. Karniadakis. “Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations”. In: *Journal of Computational Physics* 378 (2019), pp. 686–707 (cit. on p. 4).

[56] K. T. Richard Licata Piyush Mehta. “Machine-Learned HASDM Thermospheric Mass Density Model With Uncertainty Quantification”. In: *Space Weather* 20 (2022). DOI: https://doi.org/10.1029/2021SW002915 (cit. on pp. 15–17).

[57] T. G. Roberts. *Popular Orbits 101*. 2022. URL: https://aerospace.csis.org/aerospace101/earth-orbit-101/ (cit. on p. 27).

[58] H. Schaeffer et al. “Sparse dynamics for partial differential equations”. English (US). In: *Proceedings of the National Academy of Sciences of the United States of America* 110.17 (Apr. 2013), pp. 6634–6639. ISSN: 0027-8424. DOI: 10.1073/pnas.1302752110 (cit. on p. 18).

[59] M. D. Schmidt et al. “Automated refinement and inference of analytical models for metabolic networks”. In: *Physical Biology* 8.5 (Aug. 2011), p. 055011. DOI: 10.1088/1478-3975/8/5/055011. URL: https://dx.doi.org/10.1088/1478-3975/8/5/055011 (cit. on p. 18).

[60] P. Sedgwick. “Pearson’s correlation coefficient”. In: *BMJ* 345 (July 2012), e4483–e4483. DOI: 10.1136/bmj.e4483 (cit. on p. 25).

[61] N. W. Service. *Air Pressure*. URL: https://www.weather.gov/jetstream/pressure (cit. on p. 3).

[62] B. Silva. *Practical Tips - Optimization*. 2020. URL: https://pysindy.readthedocs.io/en/latest/tips.html#optimization (cit. on p. 30).

[63] H. Song and S. Hu. “Open Problems in Applications of the Kalman Filtering Algorithm”. In: *Proceedings of the 2019 International Conference on Mathematics, Big Data Analysis and Simulation and Modelling (MBDASM 2019)*. Atlantis Press, 2019/10, pp. 185–190. ISBN: 978-94-6252-811-6. DOI: 10.2991/mbdasm-19.2019.43. URL: https://doi.org/10.2991/mbdasm-19.2019.43 (cit. on p. 17).

[64] N. Srivastava et al. “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”. In: *Journal of Machine Learning Research* 15.56 (2014), pp. 1929–1958. URL: http://jmlr.org/papers/v15/srivastava14a.html (cit. on p. 16).

[65] J. N. K. Steven L. Brunton Joshua L. Proctor. “Discovering governing equations from data by sparse identification of nonlinear dynamical systems”. In: *Proceedings of the National Academy of Sciences* 113.15 (2016), pp. 3932–3937. DOI: https://doi.org/10.1073/pnas.1517384113 (cit. on pp. 18–20, 28).

[66] S. M. Stigler. “Gauss and the Invention of Least Squares”. In: *The Annals of Statistics* 9 (1981), pp. 465–474 (cit. on p. 4).

[67] M. F. Storz et al. “High accuracy satellite drag model (HASDM)”. In: *Advances in Space Research* 36.12 (2005). Space Weather, pp. 2497–2505. ISSN: 0273-1177. DOI: https://doi.org/10.1016/j.asr.2004.02.020. URL: https://www.sciencedirect.com/science/article/pii/S0273117705002048 (cit. on pp. 2, 16).

[68] S. Subramanian et al. *Adaptive Self-supervision Algorithms for Physics-informed Neural Networks*. 2022. arXiv: 2207.04084 [cs.LG] (cit. on pp. 15, 65, 70).

[69] S. Suthaharan. “Support Vector Machine”. In: *Machine Learning Models and Algorithms for Big Data Classification: Thinking with Examples for Effective Learning*. Boston, MA: Springer US, 2016, pp. 207–235. ISBN: 978-1-4899-7641-3. DOI: 10.1007/978-1-4899-7641-3_9. URL: https://doi.org/10.1007/978-1-4899-7641-3_9 (cit. on p. 21).

[70] S. Wang, S. Sankaran, and P. Perdikaris. *Respecting causality is all you need for training physics-informed neural networks*. 2022. arXiv: 2203.07404 [cs.LG] (cit. on pp. 15, 65, 70).

[71] S. Wang, Y. Teng, and P. Perdikaris. *Understanding and mitigating gradient pathologies in physics-informed neural networks*. 2020. arXiv: 2001.04536 [cs.LG] (cit. on p. 15).

[72] S. Wang, X. Yu, and P. Perdikaris. “When and why PINNs fail to train: A neural tangent kernel perspective”. In: *Journal of Computational Physics* 449 (2022), p. 110768. ISSN: 0021-9991. DOI: https://doi.org/10.1016/j.jcp.2021.110768. URL: https://www.sciencedirect.com/science/article/pii/S002199912100663X (cit. on pp. 15, 65, 70).

[73] B. Weber. *Classical Orbital Elements*. URL: https://orbital-mechanics.space/classical-orbital-elements/classical-orbital-elements.html (cit. on p. 6).

[74] B. Weeden. *2007 Chinese Anti-Satellite Test Fact Sheet*. 2010. URL: https://swfound.org/media/9550/chinese_asat_fact_sheet_updated_2012.pdf (cit. on p. 1).

[75] B. Weeden. *2009 Iridium-Cosmos Collision Fact Sheet*. 2010. URL: https://swfound.org/media/6575/swf_iridium_cosmos_collision_fact_sheet_updated_2012.pdf (cit. on p. 2).

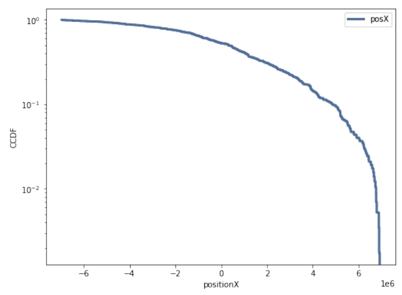
[76] D. R. Weimer. *A comparison of the JB2008 and NRLMSISE-00 neutral density models*. URL: https://ccmc.gsfc.nasa.gov/static/files/Mini-GEM-%202014-GEM-CEDAR-Weimer.pdf (cit. on p. 8).

[77] Z. Xu et al. "Recursive long short-term memory network for predicting nonlinear structural seismic response". In: *Engineering Structures* 250 (2022), p. 113406. ISSN: 0141-0296. DOI: https://doi.org/10.1016/j.engstruct.2021.113406. URL: https://www.sciencedirect.com/science/article/pii/S0141029621015133 (cit. on p. 42).

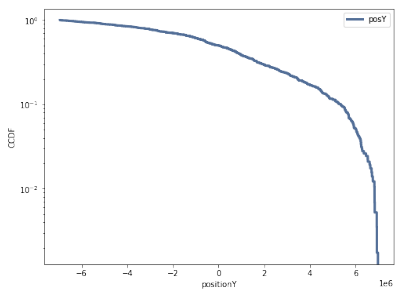
[78] L. Yunpeng et al. "Multi-step Ahead Time Series Forecasting for Different Data Patterns Based on LSTM Recurrent Neural Network". In: *2017 14th Web Information Systems and Applications Conference (WISA)*. 2017, pp. 305–310. DOI: 10.1109/WISA.2017.25 (cit. on p. 40).

I

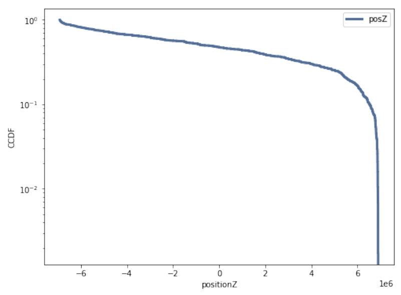
Annex 1 - CCDF Plots



(a) CCDF Position X

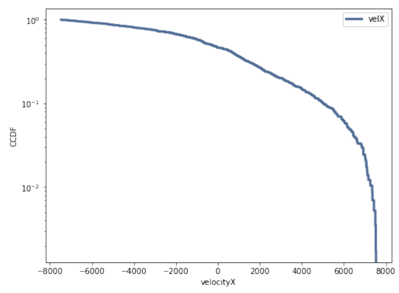


(b) CCDF Position Y

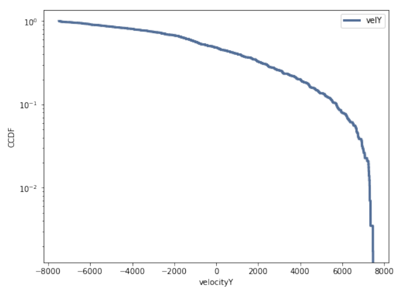


(c) CCDF Position Z

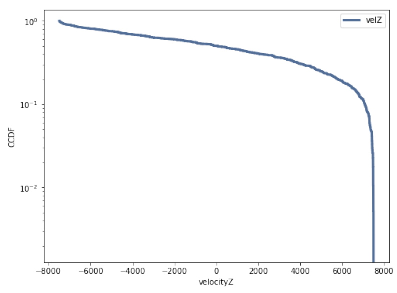
Figure I.1: CCDF Plots - Positions (Satellite 0903)



(a) CCDF Velocity X



(b) CCDF Velocity Y



(c) CCDF Velocity Z

Figure I.2: CCDF Plots - Velocities (Satellite 0903)

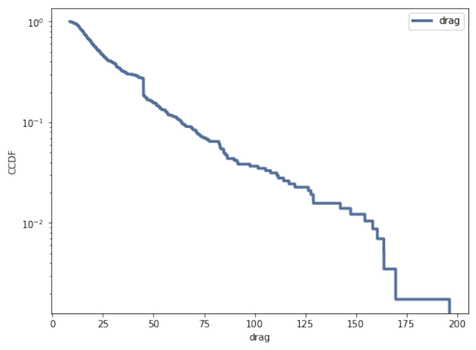
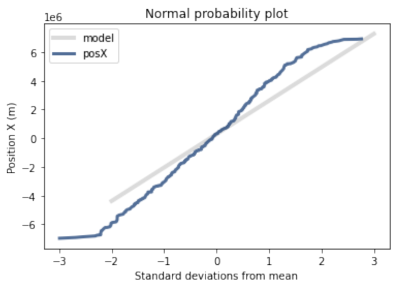


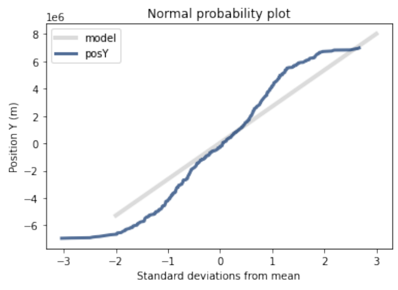
Figure I.3: CCDF Plot - Drag (Satellite 0903)

II

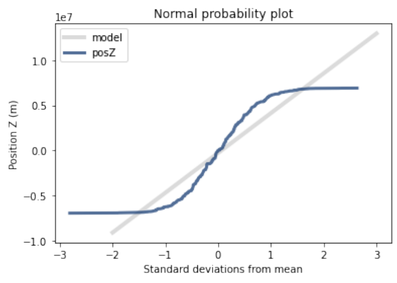
Annex 2 - Normal Probability Plots



(a) Normal Prob. Plot - Position X

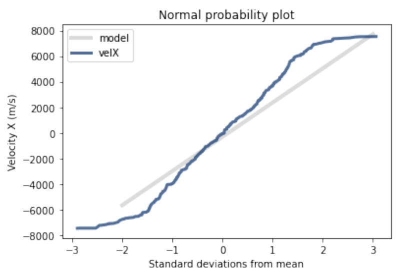


(b) Normal Prob. Plot - Position Y

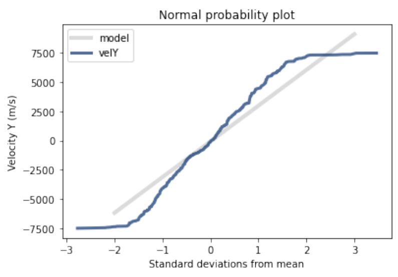


(c) Normal Prob. Plot - Position Z

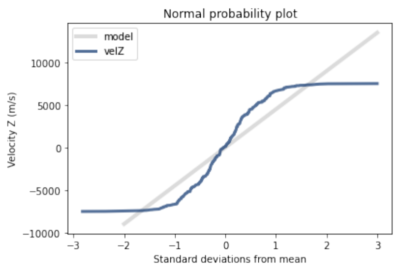
Figure II.1: Normal Probability Plots - Positions (Satellite 0903)



(a) Normal Prob. Plot - Velocity X



(b) Normal Prob. Plot - Velocity Y



(c) Normal Prob. Plot - Velocity Z

Figure II.2: Normal Probability Plots - Velocities (Satellite 0903)

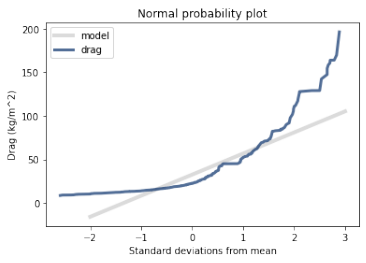


Figure II.3: Normal Probability Plot - Drag (Satellite 0903)

III

Annex 3 - Correlation Plot

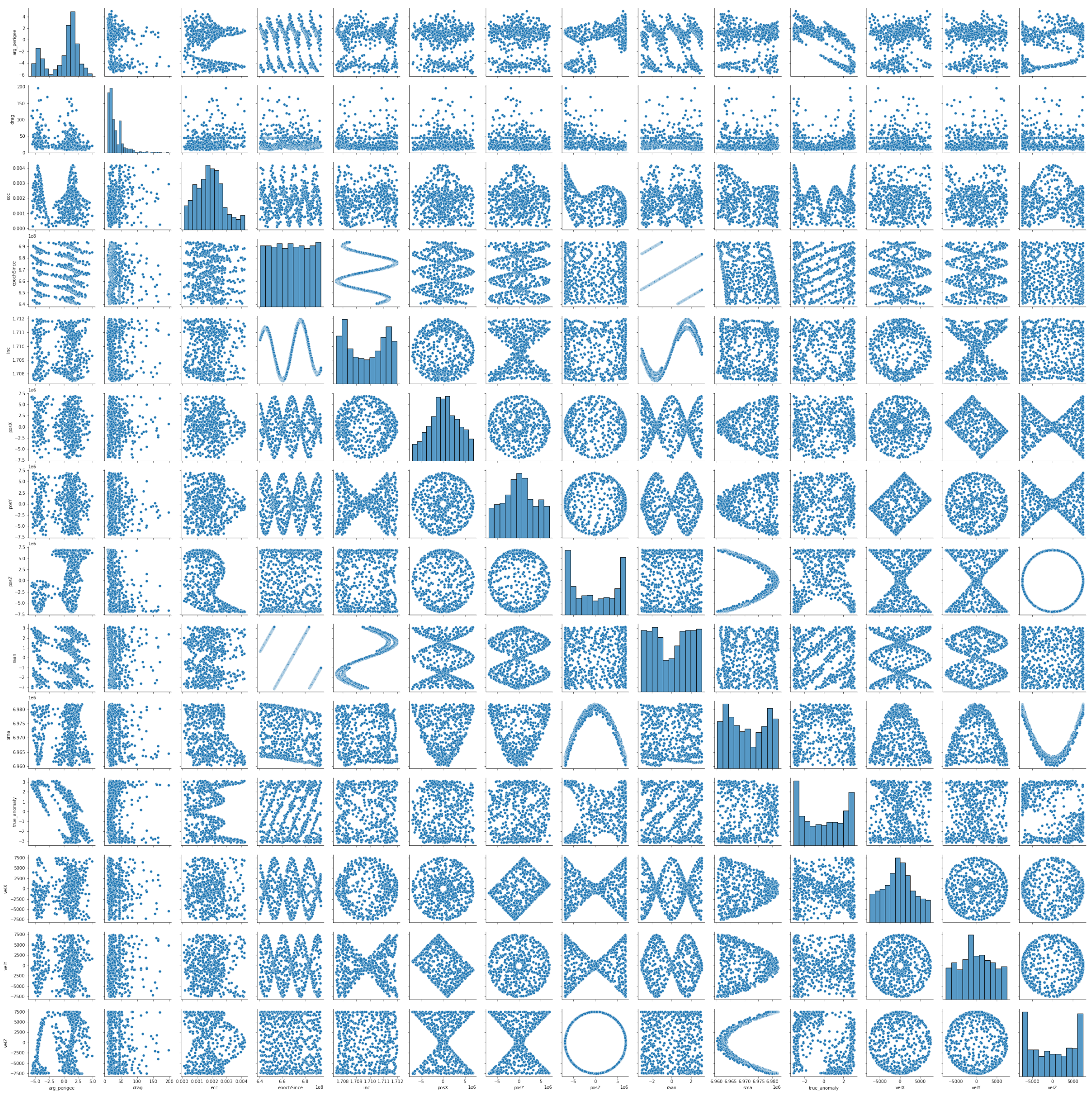


Figure III.1: Correlation Plot - Satellite ID: 0903